EXAMPLES OF FORMAL DESCRIPTIONS OF COMMON PROGRAMMING LANGUAGES

by

W. D. Maurer, C. Kinder, E. Powers, R. O. Nichols,

W. Lapinsky, L. H. Yeung, N. H. McDonald, and S. Zalewski

Memorandum No. ERL-M396

August 1973

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TABLE OF CONTENTS

INTRODUCTION

In a previous report [1] we have illustrated the application of our syntactic and semantic conventions to the formal description of several programming languages. These conventions have since been modified and extended in a number of ways, and they have themselves been thoroughly documented [2]. The present report supersedes those sections of [1] which concern themselves with formal syntax and semantics. Since the authors of the various sections of [1] have finished their studies, it was decided to start anew, rather than making modifications and extensions to [1].

Our basic goal in developing a method of formal syntactic and semantic description has been to develop such a method with the following two properties:

(a) It should be applicable to commonly used programming languages, rather than to languages invented for the purpose of illustrating the description method. This property of our method is illustrated here, by descriptions of FORTRAN, COBOL, ALGOL 60, BASIC, LISP, PDP-8 assembly language (PAL-III), and Burroughs's systems programming language (BPL).

(b) It should be amenable to proofs of correctness; that is, given a formal description in this form, one ought to be able to prove that a program written in the described language is correct. A description of how this is done has been given in a recent doctoral dissertation [3] written by a student of the first author named above.

REFERENCES

1. Maurer, W. D., Examples of algorithm verification, Memorandum M-291, Electronics Research Laboratory, University of California, Berkeley, California, 1971.

2. Maurer, W. D., Introduction to programming science, Part I: Syntax and semantics of programming languages, Memorandum M-368, Electronics Research Laboratory, University of California, Berkeley, December 1972.

3. Chang, Joon, A semantics-directed program verifier, Ph. D. Thesis, University of California, Berkeley, California, September 1973.

Research sponsored by the National Science Foundation, Grant GJ-31612.

CHAPTER TWO

A DESCRIPTION OF ALGOL 60

BY

CHARLES KINDER

Preliminaries

Letters

⟨letter⟩ ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

Logical Values

⟨logical value⟩ ::= true | false

Identifier

⟨identifier⟩ ::= ⟨letter⟩ | ⟨identifier⟩ ⟨letter⟩ | ⟨identifier⟩ ⟨digit⟩

Expression

⟨expression⟩ w::= ⟨arithmetic expression⟩ a; x::= ⟨Boolean expression⟩ b;
y::= ⟨designational expression⟩ d

⟨value⟩ $w^v(S) = a^v(S)$; $x^v(S) = b^v(S)$; $y^v(S) = d^v(S)$

⟨exit⟩ $w^x(S) = a^x(S)$; $x^x(S) = b^x(S)$; $y^x = d^x(S)$

⟨side-effect⟩ $w^s(S) = a^s(S)$; $x^s(S) = b^s(S)$; $y^s(S) = d^s(S)$

⟨environment⟩ $a^y = w^y$; $b^y = x^y$; $d^y = y^y$

⟨L-value⟩ $w^l(S) = a^l(S)$; $x^l(S) = b^l$; $y^l(S) = d^l(S)$

Numbers

⟨digit⟩ a::='0'; b::='1'; c::='2'; d::='3'; e::='4'; f::='5'; g::='6';
h::='7'; i::='8'; j::='9';
value $a^V=0$; $b^V=1$; $c^V=2$; $d^V=3$; $e^V=4$; $f^V=5$; $g^V=6$; $h^V=7$; $i^V=8$; $j^V=9$;

⟨unsigned integer⟩ x::=⟨digit⟩ u; y::=⟨unsigned integer⟩ z ⟨digit⟩ v
⟨value⟩ $x^V=u^V$; $y^V=10\ z^V+v^V$
$(\alpha \leq y^V \leq \beta)$
⟨length⟩ $x^n=1$; $y^n=z^n+1$
α,β dependent on word size of computer

⟨integer⟩ x::=⟨unsigned integer⟩ u; y:: '+' ⟨unsigned integer⟩ v;
z::='-'⟨unsigned integer⟩ w;
⟨value⟩ $x^V=u^V$; $y^V=v^V$; $z^V=-w^V$

⟨decimal fraction⟩ f::='.' ⟨unsigned integer⟩ i
⟨value⟩ $f^V=i^V/\exp(10,i^n)$

⟨exponent part⟩ e::='$_{10}$' ⟨integer⟩ i
⟨value⟩ $e^V=\exp(10,i^V)$

⟨decimal number⟩ x::=⟨unsigned integer⟩ u; y::=⟨decimal fraction⟩ f;
z::=⟨unsigned integer⟩ v ⟨decimal fraction⟩ g
⟨value⟩ $x^V=u^V$; $y^V=f^V$; $z^V=v^V+g^V$
⟨type⟩ $x^t=$ integer; $y^t=$ real; $z^t=$ real;

⟨unsigned number⟩ x::=⟨decimal number⟩ u; y::=⟨exponent part⟩ p; z::=⟨decimal number⟩ v ⟨exponent part⟩ q
⟨value⟩ $x^V=u^V$; $y^V=p^V$; $z^V=v^V\cdot q^V$

⟨type⟩ $x^t = u^t$; y^t real; z^t real;

⟨number⟩ x::=⟨unsigned number⟩ u; y::='+'⟨unsigned number⟩ v;
z::='-'⟨unsigned number⟩ w;
⟨value⟩ $x^v = u^v$; $y^v = v^v$; $z^v = -w^v$
⟨type⟩ $x^t = u^t$; $y^t = v^t$; $z^t = w^t$;

Arithmetic Expressions

⟨adding operator⟩ p::= '+'; m:: '−'

⟨binary function⟩ p^b= plus; m^b= difference

⟨unary function⟩ p^u= indent; m^u= neg;

⟨multiplying operator⟩ t::='x'; r::='/'; i::='x'

⟨binary function⟩ t^b=times; r=rdiv; i^b=idiv

⟨primary⟩ w::=⟨unsigned number⟩ u; x::=⟨variables⟩ v; y::=⟨function designator⟩ f; z::= '('⟨arithmetic exp⟩ e')'

⟨exit⟩ $w^x(S)$=normal; $x^x(S)=v^x(S)$; $y^x(S)=f^x(S)$; $z^x(S)=e^x(S)$

⟨L- value⟩ $u^l(S)$=error; $x^l(S)=v^l(S)$; $y^l(S)=$ error; $z^l(S)=$ error

⟨side effect⟩ $w^s(S)=S$; $x^s(S)=v^s(S)$; $y^s(S)=f^s(S)$; $z^s(S)=e^s(S)$

⟨value⟩ $w^v(S)=u^v$; $x^v(S)=$ if $x^x(S)\neq$normal then none else $v^v(S)$;

$y^v(S)=$ if $y^x(S)\neq$normal then none else $f^v(S)$; $z^v(S)=$if $z^x(S)\neq$ normal then none else $e^l(S)$

⟨environment⟩ $v^y=x^y$; $f^y=y^y$; $e^y=z^y$

⟨factor⟩ x::=⟨primary⟩ p; y::=⟨factor⟩ z '↑' ⟨primary⟩q;

⟨exit⟩ $x^x(S)=p^x(S)$; $y^x(S)=$ if $z^x(S)\neq$ normal then $z^x(S)$ else $q^x(z^s(S))$;

⟨side effect⟩ $x^s(S)=p^s(S)$; $y^s(S)=$if $z^x(S)\neq$normal then $z^s(S)$ else $p^s(z^s(S))$

⟨value⟩ $x^v(S)=$ if $x^x(S)\neq$ normal then none else $p^v(S)$;

$y^v(S)=$if $y^x(S)\neq$normal then none else exp $(z^v(S), q^v(z^s(S))$

⟨environment⟩ $p^y=x^y$; $z^y=y^y$; $q^y=y^y$

⟨L-value⟩ $x^l(S)=p^l(S)$; $y^l(S)=$ error

⟨term⟩ x::= ⟨factor⟩f; y::= ⟨term⟩ z ⟨multiplying operator⟩ o ⟨factor⟩ g

⟨exit⟩ $x^x(S) = f^x(S)$; $y^x(S) =$ if $z^x(S) \neq$ normal then $z^x(S)$ else $g^x(z^s(S))$

⟨side effect⟩ $x^s(S) = f^s(S)$; $y^s(S) =$ if $z^x(S) \neq$ normal then $z^s(S)$ else $g^s(z^s(S))$

⟨value⟩ $x^v(S) =$ if $x^x(S) \neq$ normal then none else $f^v(S)$; $y^v(S) =$ if $z^x(S) \neq$ normal then none else $o^b(z^v(S), g^v(z^s(S)))$

⟨environment⟩ $f^y = x^y$; $z^y = y^y$; $g^y = y^y$

⟨L-value⟩ $x^l(S) = f^l(S)$; $y^l(S) =$ error

⟨simple arithmetic expression⟩ x::= ⟨term⟩t; y::=⟨adding operator⟩ a ⟨term⟩u; z::= ⟨simple arithmetic expression⟩ s ⟨adding operator⟩ b ⟨term⟩ v

⟨exit⟩ $x^x(S) = t^x(S)$; $y^x(S) = u^x(S)$; $z^x(S) =$ if $s^x(S) \neq$ normal then $s^x(S)$ else $v^s(s^s(S))$

⟨value⟩ $x^v(S) =$ if $x^x(S) \neq$ normal then none else $t^v(S)$; $y^v(S) =$ if $y^x(S) \neq$ normal then none else $a^u(u^v(S))$; $z^v(S) =$ if $z^x(S) \neq$ normal then none else $b^b(s^v(S), v^v(s^s(S))$

⟨environment⟩ $t^y = x^y$; $a^y = y^y$; $s^y = z^y$; $v^y = z^y$

⟨L-value⟩ $x^l(S) = t^l(S)$; $y^l(S) =$ error; $z^l(S) =$ error

⟨side effect⟩ $x^s(S) = t^s(S)$; $y^s(S) = u^s(S)$; $z^s(S) =$ if $s^x(S) \neq$ normal then $s^s(S)$ else $v^s(s^s(S))$

⟨if clause⟩ i::= 'if' ⟨Boolean expression⟩ b 'then'

⟨exit⟩ $i^x(S) = b^x(S)$

⟨side effect⟩ $i^s(S) = b^s(S)$

⟨value⟩ $i^v(S) =$ if $i^x(S) \neq$ normal then none else $b^v(S)$

⟨environment⟩ $b^y = i^y$

⟨arithmetic expression⟩ x::=⟨simple arithmetic⟩ s; y::=⟨if clause⟩ i ⟨simple arithmetic expression⟩ t 'else' ⟨arithmetic expression⟩ z

⟨exit⟩ $x^X(S) = s^X(S)$; $y^X(S) =$ if $i^X(S) \neq$ normal then $i^X(S)$ else if $i^V(S)$ then $t^X(i^S(S))$ else $z^X(i^S(S))$

⟨side effect⟩ $x^S(S) = s^S(S)$; $y^S(S) =$ if $i^X(S) \neq$ normal then $i^S(S)$ else if $i^V(S)$ then $t^S(i^S(S))$ else $z^S(i^S(S))$

⟨value⟩ $x^V(S) =$ if $x^X(S) \neq$ normal then none else $s^V(S)$; $y^V(S) =$ if $y^X(S) \neq$ normal then none else if $i^V(S)$ then $t^V(i^S(S))$ else $z^V(i^S(S))$

⟨environment⟩ $s^y = x^y$; $i^y = y^y$; $t^y = y^y$; $z^y = y^y$

⟨L-value⟩ $x^l(S) = s^l(S)$; $y^l(S) =$ error

Boolean Expressions

⟨relational operator⟩ u::= '<'; v::= '≤'; w::= '='; x::= '≥';
y::= '>'; z::= '≠'

⟨binary operator⟩ u^b= less; v^b= notgreater; w^b= equal;
x^b= notless; y^b= greater; z^b=notequal

⟨relation⟩ r::= ⟨simple arithmetic expression⟩ x⟨relational operator⟩
o ⟨simple arithmetic expression⟩

⟨exit⟩ $r^x(s)$= if $x^x(S)\neq$ normal then $x^x(S)$ else $y^x(x^s(S))$

⟨side effect⟩ $r^s(S)$= if $x^x(S)\neq$ normal then $x^s(S)$ else $y^s(x^s(S))$

⟨value⟩ $r^v(S)$= if $r^x(S)\neq$ normal then none else $o^b(x^v(s)),y^v(x^s(S)))$

⟨environment⟩ x^y= r^y; y^y= r^y

⟨Boolean primary⟩ t::=⟨logical values⟩ u; w::=⟨variable⟩ v; x::= ⟨function designator⟩ f; y::= ⟨relation⟩ r; z::= '(' ⟨Boolean expression⟩b ')'

⟨exit⟩ $t^x(S)$= normal; $w^x(S)$= $v^x(S)$ $x^x(S)$= $f^x(S)$; $y^x(S)$= $r^x(S)$;
$z^x(S)$= $b^x(S)$

⟨side effect⟩ $t^s(S)$= S; $w^s(S)$= $v^s(S)$; $x^s(S)$= $f^s(S)$; $y^s(S)$= $r^s(S)$;
$z^s(S)$= $b^s(S)$

⟨value⟩ $t^v(S)\equiv u^v$; $w^v(S)$= if $w^x(S)\neq$ normal then none else $v^e(S)$;
$x^v(S)$= if $x^x(S)\neq$ normal then none else $f^v(S)$; $y^v(S)$= if $f^x(S)$
normal then none else $r^v(S)$; $z^v(S)$= if $z^x(S)\neq$ normal then none
else $b^v(S)$

⟨environment⟩ u^y= y^y; v^y= w^y; f^y= x^y; r^y= y^y; b^y= z^y

⟨L- value⟩ $t^l(S)$= error, $w^l(S)$= $v^l(S)$; $x^l(S)$= error; $y^l(S)$= error;
$z^l(S)$= error

::

⟨Boolean secondary⟩ s::=⟨Boolean primary⟩ p; t::= '¬' ⟨Boolean primary⟩ q

⟨exit⟩ $s^x(S) = p^x(S)$; $t^x(S) = q^x(S)$

⟨side effect⟩ $s^s(S) = p^s(S)$; $t^s(S) = q^s(S)$

⟨value⟩ $s^v(S) =$ if $s^x(S) \neq$ normal then none else $p^v(S)$; $t^v(S) =$ if $t^x(S) \neq$ normal then none else $q^v(S)$

⟨environment⟩ $p^y = s^y$; $q^y = t^y$

⟨L-value⟩ $s^l(S) = p^l(S)$; $t^l(S) =$ error

⟨Boolean factor⟩ x::=⟨Boolean secondary⟩ s; y::=⟨Boolean factor⟩ z '∧'

⟨Boolean secondary⟩ t

⟨exit⟩ $x^x(S) = s^x(S)$; $y^x(S) =$ if $z^x(S) \neq$ normal then $z^x(S)$ else $t^x(z^s(S))$

⟨side effect⟩ $x^s(S) = s^s(S)$; $y^s(S) =$ if $z^x(S) \neq$ normal then $z^s(S)$ else $t^s(z^s(S))$

⟨value⟩ $x^v(S) =$ if $x^x(S) \neq$ normal then none else $s^r(S)$; $y^v(S) =$ if $y^x(S) \neq$ normal then none else and $(z^v(S), t^v(z^s(S)))$

⟨environment⟩ $s^y = x^y$; $z^y = y^y$; $t^y = y^y$

⟨L-value⟩ $x^l(S) = s^l(S)$; $y^l(S) =$ error

⟨Boolean term⟩ x::=⟨Boolean factor⟩ f; y::=⟨Boolean term⟩ z '∨'

⟨Boolean factor⟩ g

⟨exit⟩ $x^x(S) = f^x(S)$; $y^x(S) =$ if $z^x(S) \neq$ normal then $z^x(S)$ else $g^x(z^s(S))$

⟨side effect⟩ $x^s(S) = f^s(S)$; $y^s(S) =$ if $z^x(S) \neq$ normal then $z^s(S)$ else $g^s(z^s(S))$

⟨value⟩ $x^v(S) =$ if $x^x(S) \neq$ normal then none else $f^v(S)$; $y^v(S) =$ if $y^x(S)$ normal then none else or $(z^v(S), g^v(z^s(S)))$

⟨environment⟩ $f^y = x^y$; $z^y = y^y$; $g^y = y^y$

⟨L-value⟩ $x^l(S) = f^l(S)$; $y^l(S) =$ error

⟨implication⟩ x::= ⟨Boolean term⟩ t; y::= ⟨implication⟩ z '⊃'

⟨Boolean term⟩ u

⟨exit⟩ $x^x(S) = t^x(S)$; $y^x(S) =$ if $z^x(S)$ normal then $z^x(S)$ else $u^x(z^s(S))$

⟨side effect⟩ $x^s(S) = t^s(S)$; $y^s(S) =$ if $z^x(S)$ normal then $z^s(S)$ else $u^s(z^s(S))$

⟨value⟩ $x^v(S) =$ if $x^x(S) \neq$ normal then none else $t^v(S) =$ if $y^x(S) \neq$ normal then none else implies $(z^v(S), u^v(z^s(S)))$

⟨environment⟩ $t^y = x^y$; $z^y = y^y$; $u^y = y^y$

⟨L-value⟩ $x^l(S) = t^l(S)$; $y^l(S) =$ error

⟨simple Boolean⟩ x::= ⟨implication i; y::= ⟨simple Boolean⟩ z '≡'

⟨implication⟩ j

⟨exit⟩ $x^x(S) = i^x(S)$; $y^x(S) =$ if $z^x(S) \neq$ normal then $z^x(S)$ else $j^x(i^s(S))$

⟨side effect⟩ $x^s(S) = i^s(S)$; $y^s(S) =$ if $z^x(S) \neq$ normal then $z^s(S)$ else $j^s(i^s(S))$

⟨value⟩ $x^v(S) =$ if $x^x(S) \neq$ normal then none else $i^v(S)$; $y^v(S) =$ if $y^x(S) \neq$ normal then none else equivalence $(z^v(S), j^v(z^s(S)))$

⟨environment⟩ $i^y = x^y$; $z^y = y^y$; $j^y = y^y$

⟨L-value⟩ $x^l(S) = i^l(S)$; $y^l(S) =$ error

⟨Boolean expression⟩ x::= ⟨simple Boolean⟩ s; y::= ⟨if clause⟩ i

⟨simple Boolean⟩ t 'else' ⟨Boolean expression⟩ z

⟨exit⟩ $x^X(S) = s^X(S)$; $y^X(S) =$ if $i^X(S) \neq$ normal then $i^X(S)$ else if $i^V(S)$ then $t^X(i^S(S))$ else $z^X(i^S(S))$

⟨side effect⟩ $x^S(S) = s^S(S)$; $y^S(S) =$ if $i^S(S)$ normal then $i^S(S)$ else if $i^V(S)$ then $t^S(i^S(S))$ else $z^S(i^S(S))$

⟨value⟩ $x^V(S) =$ if $x^X(S) \neq$ normal then none else $s^V(S)$; $y^V(S) =$ if $y^X(S) \neq$ normal then none else if $i^V(S)$ then $t^V(i^S(S))$ else $z^V(i^S(S))$

⟨environment⟩ $s^y = x^y$; $i^y = y^y$; $t^y = y^y$; $z^y = y^y$

⟨L-value⟩ $x^l(S) = s^l(S)$; $y^l(S) =$ error

Designational Expression

⟨label⟩ x::= ⟨identifier⟩ i; y::= ⟨unsigned integer⟩ u

⟨switched identifier⟩ ::=⟨identifier⟩

⟨switched designator⟩ d::= ⟨switch identifier⟩ s '[' ⟨subscript expression⟩ e ']'

⟨label function⟩ ^{a}

⟨environment⟩ ^{y} $s^y = d^y$

⟨exit⟩ $d^x(S) =$ if $e^x(S) \neq$ normal then $d^a(d^y(s, e^v(S), \text{label}))$ $(d^y(s, \text{switch range}) \geq e^v(S) \geq 1)$

⟨side effect⟩ $d^s(S) = e^s(S)$

⟨simple desgnational expression⟩ x::= ⟨label⟩ a; y::= ⟨switch designator⟩ d; z::= '(' ⟨designational expression⟩ e ')'

⟨label funcion⟩ ^{a}

$d^a = y^a$; $e^a = z^a$

⟨exit index⟩ $x^x(S) \equiv x^a(a)$; $y^x(S) = d^x(S)$; $z^x(S) = e^x(S)$

⟨side effect⟩ $x^s(S) = s$; $y^s(S) = d^s(S)$;

⟨environment⟩ $a^y = x^y$; $d^y = y^y$; $e^y = z^y$

⟨designational expression⟩ x::= ⟨simple designational expression⟩ d; y::= ⟨if clause⟩ i ⟨simple designational expression⟩ e 'else' ⟨designational expression⟩ z

⟨label function⟩ ^{a} $d^a = x^a$; $i^a = y^a$; $e^a = y^a$; $z^a = y^a$

⟨exit index⟩ $x^x(S) = d^x(S)$; $y^x(S) =$ if $i^x(S)$ normal then $i^x(S)$ else if $i^v(S)$ then $e^x(i^s(S))$ else $z^x(i^s(S))$

⟨side effect⟩ $x^s(S) = d^s(S)$; $y^s(S) =$ if $i^x(S) \neq$ normal then $i^s(S)$ else if $i^v(S)$ then $e^s(i^s(S))$ else $z^s(i^s(S))$

⟨environment⟩ $d^y = x^y$; $i^y = y^y$; $e^y = y^y$; $z^y = y^y$

⟨L-value⟩ $x^l(S) =$ error; $y^l(S) =$ error

Variables

⟨variable identifier⟩ ::= ⟨identifier⟩

⟨array identifier⟩ ::= ⟨identifier⟩

⟨simple variable⟩ v::=⟨variable identifier⟩ i

⟨value⟩ $v^v(S)$= if $v^y(v,pcm)$= name then (u,e,x,) else S(v)

⟨side effect⟩ $v^s(S)$= if $v^y(v,pcm)$ = name then e(S) where S(v)= (u,e,x)
else S

⟨exit⟩ $v^x(S)$= if $v^y(v,pcm)$= name then e(S), where S (v)= (u,e,x)
else S

⟨L-value⟩ $v^l(S)$= if $v^y(v,pcm)$=name then u(S) where S(v)= (u,e,x)
else v
($v^y(v,rank)$= o)

⟨subscripted expression⟩ s::= ⟨arithmetic expression⟩ e

⟨value⟩ $s^v(S)$= if $s^x(S)\neq$ normal then none else $e^v(S)$
($e^t(S)\equiv$ integer)

⟨exit⟩ $s^x(S)$= $e^x(S)$

⟨side effect⟩ $s^s(S)$= $e^s(S)$

⟨environment⟩ e^y=s^y

⟨subscript list⟩ x::= ⟨subscript expression⟩ s; y::= ⟨subscript list⟩
z ',' ⟨subscript expression⟩ t

⟨n-tuple of values⟩ $x^v(S)$= $s^v(S)$; $y^v(S)$=if $y^x(S)\neq$normal then ...
none else concat ($z^v(S)$,',',$t^v(S)$)

⟨exit⟩ $x^x(S)$=$s^x(S)$; $y^x(S)$=if $z^x(S)\neq$normal then $z^x(S)$ else $t^x(z(S))$

⟨side effect⟩ $x^s(S)$= $s^s(S)$; $y^s(S)$= if $z^x(S)\neq$ normal then $z^s(S)$
else $t^s(z^s(S))$

⟨number of subscripts⟩ s^j= 1; y^j= z^j+1

⟨environment⟩ s^y= x^y; z^y= y^y; t^y= y^y;

⟨subscripted variable⟩ s::= ⟨array identifier⟩ a '[' ⟨subscript list⟩ t ']'

⟨L-value⟩ $s^l(S)$ = **if** $s^y(a,pcm)$ = **name then** $u(S)$ where $S(s) = (u,e,x)$ **else if** $s^x(S) \neq$ **normal then none else concat** $(a,'[',t^v(S),']')$

⟨side effect⟩ $s^s(S)$ = **if** $s^y(a,pcm)$ = **name then** $e(S)$ where $S(s) = (u,e,x)$ **else** $t^s(S)$

⟨value⟩ $s^v(S)$ = **if** $s^y(a,pcm)$ = **name then** (u,e,x) **else if** $s^x(S)$ = **normal then none else** $S(s^l(S))$

⟨number of subscripts⟩ $s^j(S) = t^j(S)$

⟨environment⟩ $a^y = s^y$; $t^y = s^y$

$(s^y(a,rank) = t^j)$

⟨exit⟩ $s^x(S)$ = **if** $s^y(a,pcm)$ = **name then** $x(S)$ where $S(s) = (u,e,x)$ **else** $t^x(S)$

⟨variable⟩ x::= ⟨simple variable⟩ v; y::= ⟨subscripted bariable⟩ s

⟨value⟩ $x^v(S) = v^v(S)$; $y^v(S) = s^v(S)$

⟨exit⟩ $x^x(S) = v^x(S)$; $y^x(S) = s^x(S)$

⟨environment⟩ $v^y = x^y$; $s^y = y^y$

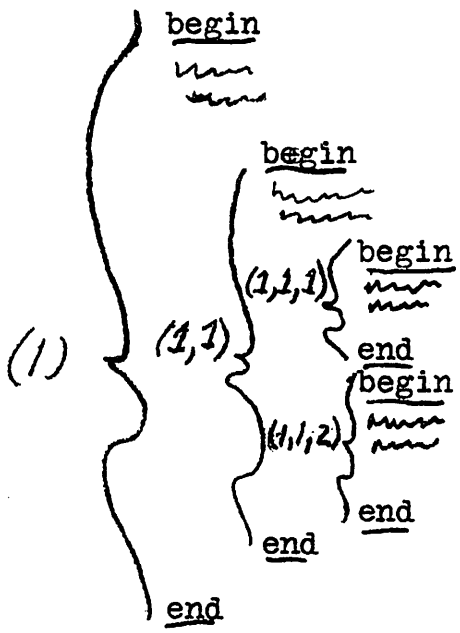
⟨side effect⟩ $x^s(S) = v^s(S)$; $y^s(S) = s^s(S)$

⟨L-value⟩ $x^l = v^l$; $y^l = u^l$

A Prelude to My Block Structure Discription

A arrangment of Algol blocks within a program may be looked upon as a set of sequences. My notation will use the notation for blocks as exemplified by the structure of the following simple program structure.

Example program structure of a block:



For each block there is a block index in the form of a ordered vector $(l_1,l_2,\ldots$ n is the level or depth of the block within the program. Note that the block with index $(l_1,l_2,\ldots l_n)$ is a subset of all blocks with index $(l_1,\ldots l_i)$ $1 \leq i < n$. In my notation I will use $(l_1,\ldots l_n) \subset (l_1,\ldots l_i)$ to mean that the block with index $(l_1,\ldots l_n)$ is contained in the block with index $(l_1,\ldots l_i)$. For example in the above program $(1,1,1) \subset (1,1) \subset (1)$ but $(1,1,1) \not\subset (1,1,2)$. For reasons that will be apparent later, I will give each statement within a block the attribute of the block index. Under this procedure, if a block A contains block B, block B as a whole (being a statement of block A) will have as an attribute the index of block A. The statements within B will have the attributes of the index of B. If the index

of A was $(l_1,...l_k)$ then the index of B would be $(l_1,...l_k,l_{k+1})$ where $(l_1,...l_{k+1}) \subseteq (l_1,...l_k)$. Under normal block operations, a statement within a block A may transfer to a statement in any block B which contains the block A. (or index block A $\subseteq$ index block B). Each time one enters a block A from a block B, one has to change the state vector S so that it localizes all the variables declared in block A. One has to also stack all the variables in block B whose names conflict with the variable names declared in block A. Upon exit from block B to block C (where block B $\subseteq$ block A) one has to change the state vector S to S'so that S' does not contain any of the variables of the blocks whose index is $\subseteq$ index C. So if the index of B was $(l_1,...l_n)$ and the index of C was $(l_1,...l_i)$ where $i \leq n$, then all variables in S whose indices are $(l_1,...l_{i+1})...$ $(l_1,...l_n)$ should be removed. If the transfer is a <u>recursive</u> transfer in which a statement within block B transfers to the head of block C, where index B $\subseteq$ index C, then all variables whose block index is equal to that of C should be put on stacks. Otherwise the variables whose indices are $(l_1...l_{i+1})...(l_1,...l_n)$ should be removed.

The attributes which I use in my notation for implimenting this block logic are the following: (a)block index (b)block count (c) block index function, and (d)block statement function. The block index is an ordered vector quanity $(l_1,l_2,...l_n)$ which indicates the block level. This is initialized at the program level. It is modified in the compound tail section. The block count is used to modify the block index. It is intialized at zero at the beginning of every block. The block index function is synthesized in the compound tail section. This associates with every statement index a block level. This is used in determining the effect of the exit from a block.

By knowing the block index of the statement exited to from the block, one knows how many variables to unstack and remove from the domain of the state vector. The block statement function is also synthesized in the compound tail section of the Algol grammar. This function associates only the indices of statements which are block heads with the block index of the block that it heads.

Compound Statements and Blocks

⟨unlabelled basic statement⟩ w::=⟨assignment statement⟩ a;
x::=⟨go to statement⟩ g; y::=⟨dummy statement⟩ d; z::=
⟨procedure statement⟩ p
⟨effect⟩ $w^e(S)= a^e(S)$; $x^e(S)= g^e(S)$; $y^e(S)= d^e(S)$; $z^e(S)= p^e(S)$
⟨statement index⟩ $a^i= w^i$; $p^i= z^i$
⟨exit⟩ $w^x(S)= a^x(S)$; $x^x(S)= g^x(S)$; $y^x(S)= d^x(S)$; $z^x(S)= p^x(S)$
⟨label function⟩ $g^a= x^a$; $p^a= z^a$
⟨environment⟩ $a^y= w^y$; $g^y= x^y$; $d^y= y^y$; $p^y= z^y$

⟨basic statement⟩ x::=⟨unlabelled basic statement⟩ u; y::=⟨label⟩
a ':' ⟨basic statement⟩ z
⟨effect⟩ $x^e(S)=u^e(S)$; $y^e(S)= z^s(S)$
⟨exit⟩ $x^x(S)=u^x(S)$; $y^x(S)= z^x(S)$
⟨statement index⟩ $u^i=x^i$; $z^i= y^i$
⟨label function⟩ $u^a= x^a$; $z^a= y^a$
⟨local label function⟩ $u^b= \emptyset$; $y^b= \{(a,y^i)\} \cup z^b$
(y^b is single valued)
⟨environment⟩ $u^y= x^y$; $z^y= y^y$

⟨unconditional statement⟩ u::=⟨basic statement⟩ a; x::=
⟨compound statement⟩ c; y::=⟨block⟩ b
⟨effect⟩ $u^e(S)= a^e(S)$; $x^e(S)= c^e(S)$; $y^e(S)= b^e(S)$
⟨exit⟩ $u^x(S)=a^x(S)$; $x^x(S)=c^x(S)$; $y^x(S)=b^x(S)$
⟨statement index⟩ $a^i= u^i$; $c^i= x^i$; $b^i= y^i$
⟨label function⟩ $a^a= u^a$; $c^a= x^a$; $b^a= y^a$

⟨environment⟩ $a^y = u^y$; $c^y = x^y$; $b^y = y^y$

⟨local label function⟩ $u^b = a^b$; $x^b = c^b$; $y^b = b^b$

⟨number of statements⟩ $u^n = 1$; $x^n = c^n$; $y^n = b^n$

⟨block determination⟩ $u^h =$ none; $x^h = c^h$; $y^h =$ block

⟨block index⟩ $a^j = u^j$; $c^j = x^j$; $b^j = y^j$

⟨block count⟩ $c^k = x^k$; $b^k = y^k$

⟨block synthesized index⟩^{s} $c^s = x^s$; $b^s = y^s$

⟨statement⟩ x::= ⟨unconditional statement⟩ u; y::= ⟨conditional statement⟩ c; z::= ⟨for statement⟩ f;

⟨effect⟩ $x^e(S) = u^e(S)$; $y^e(S) = c^e(S)$; $z^e(S) = f^e(S)$

⟨exit⟩ $x^x(S) = u^x(S)$; $y^x(S) = c^x(S)$; $z^x(S) = f^x(S)$

⟨statement index⟩ $u^i = x^i$; $c^i = y^i$; $f^i = z^i$

⟨label function⟩ $u^a = x^a$; $c^a = y^a$; $f^a = z^a$

⟨environment⟩ $u^y = x^y$; $c^y = y^y$; $f^y = z^y$

⟨local label function⟩ $x^b = u^b$; $y^b = c^b$; $z^b = f^b$

⟨number of statements⟩ $x^n = u^n$; $y^n = c^n$; $z^n = f^n$

⟨block determination⟩ $x^h = u^h$; $y^h = c^h$; $z^h = f^h$

⟨block index⟩ $u^j = x^j$; $c^j = y^j$; $f^j = z^j$

⟨block count⟩ $u^k = x^k$; $c^k = y^k$; $f^k = z^k$

⟨block synthesized index⟩^{s} $u^s = x^s$; $c^s = y^s$; $f^s = z^s$

⟨compound tail⟩ t::= ⟨statement⟩ r 'end'; u::= ⟨statement⟩ s ';' ⟨compound tail⟩ v

⟨effect⟩ $t^e(S) = r^e(S)$; $u^e(S) = v^e(s^e(S))$

⟨number of statements⟩ $t^n = r^n$; $u^n = s^n + v^n$

⟨statement index⟩ $r^i = t^i$; $s^i = u^i$; $v^i = u^i + s^n$

⟨single-step function⟩ $t^r((S,1)) = (r^e(S), r^x(S))$;

⟨single-step function⟩^{r}

[cont] $u^r((S,1)) = (s^e(S), s^x(S))$, $y^r((S,k)) = v^r((S,k))$ for $s^n + 1 \leq k \leq u^n$

⟨environment⟩ $r^y = t^y$; $s^y = u^y$; $v^y = u^y$

⟨label function⟩ $r^a = t^a$; $s^a = u^a$; $v^a = u^a$

⟨local label function⟩ $t^b = r^b$; $u^b = s^b \cup v^b$

⟨block index function⟩ $t^q = \{(r^i, t^j)\}$; $u^q = \{(s^i, u^j)\} \cup v^q$

⟨block index⟩ ^{j}

⟨block determination⟩ ^{h}

⟨block count⟩ $t^k =$ if $r^h =$ block then $t^k + 1$ else t^k; $u^k =$ if $s^h =$ block then $u^k + 1$ else t^k

⟨block, up-count⟩ $t^c = t^k$

⟨block synthesized index⟩^{s}

$r^s =$ if $r^h =$ block then $(l_1, l_2, \ldots l_n, l_{n+1})$ where $t^j = (l_1 \ldots l_n)$

[note t^j may be (nil)] and $l_{n+1} = t^k$ else t^j;

$s^s =$ if $s^h =$ block then $(l_1, \ldots l_{n+1})$ where $u^j = (l_1, \ldots l_n)$

[u^j may be (nil)] and $l_{n+1} = u^k$ else u^j;

⟨block statement number function⟩ $t^p =$ if $r^h =$ block then $\{(r^i, r^s)\}$ else none; $u^p =$ if $s^h =$ block then $\{(s^i, s^s)\} \cup v^p$ else v^p

⟨block count⟩ ^{k}

$r^k =$ if $r^h =$ block then 0 (zero) else none; $s^k =$ if $s^h =$ block then 0 (zero) else none.

⟨block index⟩^{j}

$r^i = t^s$; $s^j = u^s$

⟨block head⟩ x::= 'begin' ⟨declaration⟩ d; y::=⟨block head⟩ z ';'

⟨decalration⟩ e

⟨local environment⟩ $x^z = d^z$; $y^z = z^z \cup e^z$

⟨local variable set⟩ $x^o = d^o$

⟨unlabelled block⟩ c::= ⟨block head⟩ b ';' ⟨compound tail⟩ s

⟨statement index⟩ $s^i = c^i$

⟨number of statements⟩ $c^n = s^n$

⟨label function⟩ $s^a = c^a$

⟨environment⟩ $s^y = c^y$

⟨local environment⟩ $c^z = b^z$

$s^y = c^z$

⟨block index⟩ $s^j = c^j$; $b^j = c^j$

⟨block count⟩ $s^k = c^k$

⟨block index function⟩ $c^q = s^q$

⟨block statement number function⟩ $c^p = s^p$

⟨local variable set⟩ $c^o = b^o$

⟨local label function⟩^{b}

$c^a = s^b$

⟨block synthesized index⟩ $s^s = c^s$

⟨initial effect⟩ $c^m(S, c^o) = S'$ where if $(v = x) \in S$ (v a variable and x some value for that variable) and if name (v) = name(v') for some $v' \in c^o$ then $(v' = \sqrt{}(\text{undefined})) \in S'$ and $(v = x) \notin S'$. Futhermore $S'(\text{STACK}(v)) =$ if $S(\text{STACK}(v)) = (y_1, y_2, \ldots y_n)$ then $(y, y_1, \ldots y_n)$ else (y) where $y = (m, (v = x))$ and $m = c^y(v, \text{var})$ [or the block index of variable v]. This is done for all $v' \in c^o$ whose names conflict with those in S. $S' = S$ for all $(z = x) \in S$ such that the name (z) ≠ name (v') for all $v' \in c^o$. Every variable name v is given a stack vector whose value is S(STACK(v)). [this value may be undefined if the variable name does not have any conflicting usage.]

⟨single step function⟩^{r}

⟨generalized affect⟩ $c^g((S',k)) =$ *if* $c^i \leq k \leq c^i + s^n$ *the* $c^g(S^r((S,k)))$ *else* (S,k)

⟨intermediate effect⟩ $c^m(S') = S''$, where $(S'',k'') = c^g((S',c^i))$ for some k''

⟨exit⟩ $c^x(S') = k''$, where $(S'',k'') = c^g((S',k))$ for some S''

⟨exit effect⟩ $c^f(S'') = S'''$ where S''' will be determined as follows; Let $c^p(k'') = (l_1,\ldots l_k)$. and $c^q(k'') = (l_1,\ldots l_k, l_{k+1})$ or $c^p(k'')$ (depending on whether the exit index is respectively the beginning of a block ar not the beginning of a block.) Under this notation $c^p(k) = (l_1,\ldots l_n)$ where $(l_1,\ldots l_n) \subset (l_1,\ldots l_k)$ or it is an error. *If* $c^q(k'') \neq c^p(k'')$ and $(l_1,\ldots l_n) \subseteq (l_1,\ldots l_{k+1})$ *then* the exit is recursive. In this case the stacks of all variables are first popped. Let $S''(STACK(v)) = y_1 = (m,(v=x))$ and let p_1 be the projection function of the first variable to the space in which y_1 is in. Therfore $p_1(y_1) = m$. The heads of the stacks are popped recursiveley until one finds an element y_i in each stack such that $p(y_1) \geq c^q(k'')$. *or* until the stack is empty (which ever comes first). If one finds a y_1 $p(y_1) \geq c^q(k'')$, he then puts it back on the stack for that particular variable.

IF $[c^q(k'') \neq c^p(k'')$ and $(l_1,\ldots l_n) \not\subseteq (l_1,\ldots l_{k+1}) = c^q(k'')]$ *or if* $c^g(k'') = c^p(k'')$ *then* the above procedure of popping the variables of each stack is again carried out until one finds a stack element y_1, or until one empties the stack for each variable stack. In this case, the element y_1, if found, is discarded instead of restacked. All elements of S'' of the form $(v=x)$ whose variable v has a block index $m \neq c^p(k)$ must be taken out of S''. The block index of variable v is equal to $c^y(v,var)$ S''' may finally be seen to come from S'' by first modifying each element $z = STACK(v)$ for every variable

v such that $S'''(STACK(v))$ is given as shown above. Futhermore $S'''(z)=S''(z)$ for all z whose block index $c^y(z,var) \geq c^p(k'')$

⟨effect⟩ $c^e(S)=S'''$ as described above

⟨unlabelled compound⟩ c::= 'begin' ⟨compound tail⟩ s

⟨single step function⟩^{r}

⟨generalized effect⟩ $c^g((S,k))=$ **if** $c^i \leq k \leq c^i + s^n$ **then** $c^g(S^r((S,k)))$ **else** (S,k)

⟨effect⟩ $c^e(S)=S'$, where $(S',k')=c^g((S,c^i))$ for some k'

⟨exit⟩ $c^x(S)=k'$, where $(S',k')=c^g((S,c^i))$ for some S'

⟨statement index⟩

$s^i=c^i$

⟨number of statements⟩ $c^n=s^n$

⟨label function⟩ $s^a=c^a$

⟨environment⟩ $s^y=c^y$

⟨local label function⟩^{b}

$c^a=s^b$

⟨block synthesized index⟩ $s^s=c^s$

⟨block index⟩ $s^j=c^j$

⟨block count⟩ $s^k=c^k$

⟨block up-count⟩ $c^c=s^c$

⟨block determination⟩ $c^h=$ **if** $c^c=0$ **then** none **else** block.

⟨block index function⟩ $c^q=s^q$

⟨block statement number function⟩ $c^p=s^p$

⟨compound statement⟩ x::=⟨unlabelled compound⟩ u; y::=⟨label⟩ a ':'

⟨unlabelled compound⟩ z

⟨effect⟩ $x^e(S) = u^e(S)$; $y^e(S) = z^e(S)$

⟨exit⟩ $x^x(S) = u^x(S)$; $y^x(S) = z^x(S)$

⟨number of statements⟩ $x^n = u^n$; $y^n = z^n$

⟨label function⟩ $u^a = x^a$; $z^a = y^a$

⟨local label function⟩ $u^b = \emptyset$; $y^b = \{(a, y^i)\} \cup z^b$

⟨block synthesized index⟩ $u^s = x^s$; $z^s = y^s$

⟨statement index⟩ $u^i = x^i$; $z^i = y^i$

⟨environment⟩ $u^y = x^y$; $z^y = y^y$

⟨block index⟩ $u^j = x^j$; $z^j = y^j$

⟨block count⟩ $u^k = x^k$; $z^k = y^k$

⟨block determination⟩ $x^h = u^h$; $y^h = z^h$

⟨block index function⟩ $x^q = u^q$; $y^q = z^q$

⟨block statement number function⟩ $x^p = u^p$; $y^p = z^p$

⟨block⟩ $\underline{x} ::= ⟨$unlablelled block$⟩ \underline{u}$; $\underline{y} ::= ⟨$label$⟩ \underline{a}$ ':' ⟨unlabelled block⟩ $\underline{z}$

⟨effect⟩ $x^e(S) = u^e(S)$; $y^e(S) = z^e(S)$

⟨exit⟩ $x^x(S) = u^x(S)$; $y^x(S) = z^x(S)$

⟨number of statements⟩ $x^n = u^n$; $y^n = z^n$

⟨label function⟩ $u^a = x^a$; $z^a = y^a$

⟨statement index⟩ $u^i = x^i$; $z^i = y^i$

⟨environment⟩ $u^y = x^y$; $z^y = y^y$

⟨local label funciton⟩ $u^b = \emptyset$; $y^b = \{(a, y^i)\} \cup z^b$

⟨block index⟩ $u^j = x^j$; $z^j = y^j$

⟨block count⟩ $u^k = x^k$; $z^k = y^k$

⟨block index function⟩ $x^q = u^q$; $z^q = y^q$

⟨block statement number function⟩ $x^p = u^p$; $z^p = y^p$

⟨block synthesized index⟩ $u^s = x^s$; $z^s = y^s$

⟨program⟩ p::=⟨block⟩ b; q::=⟨compound statement⟩ c

⟨effect⟩ $p^e(S) = b^e(S)$; $q^e(S) = c^e(S)$

⟨exit⟩ $p^x(S) = b^x(S)$; $q^x(S) = c^x(S)$

⟨number of statements⟩ $p^n = b^n$; $q^n = c^n$

⟨block synthesized index⟩^{s} $b^s = (1)$; $c^s = (nil)$

⟨block count⟩^{k}

$b^k = 0$ (zero); $c^k = 0$ (zero)

Assignment Statements

⟨left part⟩ x::= ⟨variable⟩ y ':=' ;y::=⟨procedure identifier⟩ p ':=';
⟨L-value⟩ $x^l(S)= v^l(S)$

⟨environment⟩ $x^y= v^y$; $y^y= p^y$

⟨side effect⟩ $x^s(S)= v^s(S)$; $y^s(S)= p^s(S)$

⟨exit⟩ $x^x(S)= v^x(S)$; $y^x(S)= p^x(S)$

⟨left part list⟩ x::=⟨left part⟩ p; y::=⟨left part list⟩ z
⟨left part⟩ q

⟨environment⟩ ^{y} $p^y= x^y$; $q^y= y^y$

$(y^y(z,\underset{\sim}{type})= (y^y(q, \underset{\sim}{type}))$

⟨set of L-values⟩ $x^z(S)= p^l(S)$; $y^z(S)= z^z(S) \cup \{q^l(z^s(S))\}$

⟨exit⟩ $x^x(S)= p^x(S)$; $y^x(S)=$ if $q^x(S) \neq$ normal then $q^x(S)$
else $z^x(q^s(S))$

⟨side effect⟩ $x^s(S)= p^s(S)$; $y^x(S)=$ if $q^x(S) \neq$ normal then
$q^s(S)$ else $z^s(q^s(S))$

⟨assignment statement⟩ x::=⟨left part list⟩ p ⟨arithmetic expression⟩
a; y::=⟨left part list⟩ q ⟨Boolean expression⟩ b

⟨exit⟩ $x^x(S)=$ if $p^x(S) \neq$ normal then $p^x(S)$ else if $a^x(p^s(S)) \neq$
normal then $a^x(p^x(S))$ else x^i+1; $y^x(S)=$ if $q^x(S) \neq$ normal then $q^x(S)$
else if $b^x(q^s(S)) \neq$ normal then $b^x(q^s(S))$ else y^i+1

⟨statement index⟩ ^{i}

⟨effect⟩ $x^e(S)=$ if $p^x(S) \neq$ normal then $p^s(S)$ else if $a^x(p^s(S)) \neq$
normal then $a^s(p^s(S))$ else S', where $S'(z)=$apply $(ntcf(p^t,a^t),$
$a^v(p^s(S)))$ for $z \in p^z(S)$, $S'(z)= S''(z)$ for $z \notin p^z(S)$, where
$S''= a^s(p^s(S))$; $y^e(S)=$if $q^x(S) \neq$ normal then $q^s(S)$ else if
$b^x(q^s(S))$ normal then $b^s(q^s(S))$ else S' where $S'(z)=$ apply

$(ntcf(q^t, b^t),\ b^v(q^s(S)))$ for $z \in q^z(S)$, $S'(z) = S''(z)$ for $z \notin q^z(S)$, where $S'' = b^s(q^s(S))$

$\langle type \rangle^t$

$\langle set\ of\ L\text{-}value \rangle^z$

Dummy Statement

⟨dummy statement⟩ x::= ⟨empty⟩ e

⟨effect⟩ $x^e(S) = S$;

⟨exit⟩ $x^x(S) = x^i + 1$

⟨statement index⟩ ^{i}

Go To Statement

⟨go to statement⟩ g::= 'go to' ⟨designational expression⟩ e

⟨label function⟩ $e^a = g^a$

⟨effect⟩ $g^e(e^s(S) = e^s(S))$

⟨exit index⟩ $g^x(e^s(S)) = e^x(e^s(S))$

⟨environment⟩ $e^y = g^y$

Conditional Statements

⟨if statement⟩ i::= ⟨if clause⟩ c ⟨unconditional statement⟩ u

⟨effect⟩ if $c^v(S)$ then $u^e(c^s(S))$ else

⟨exit⟩ $i^x(S)$= if $c^v(S)$ then $u^x(c^s(S))$ else $c^x(S)$

⟨environment⟩ $c^y = i^y$; $u^y = i^y$

⟨label function⟩ $u^a = i^a$

⟨number of statements⟩ $i^n = u^n$

⟨block determination⟩ i^h= if u^h= block then block else none

⟨block index⟩ $u^j = i^j$

⟨block synthesized index⟩ $u^s = c^s$

⟨block count⟩ $u^k = i^k$

⟨true value⟩ $i^u(S) = c^v(S)$

⟨escape value⟩ $i^l(S) = c^s(S)$

⟨exit value⟩ $i^m(S) = c^x(S)$

⟨conditional statement⟩ w::= ⟨if statement⟩ a; x::=⟨if statement⟩ b 'else' ⟨statement⟩ s; y::= ⟨if clause⟩ i ⟨for statement⟩ f; z::= ⟨label⟩ a ':' ⟨conditional statement⟩ c

⟨true value⟩^{u}

⟨escape value⟩^{l}

⟨exit value⟩^{m}

⟨effect⟩ $w^e(S) = a^s(S)$; $x^e(S)$= if $b^u(S)$ then $b^e(S)$ else if $b^m(S) \neq$ normal then $b^l(S)$ else $s^e(b^l(S))$; $y^e(S)$= if $i^v(S)$ then $f^e(i^s(S))$ else $i^s(S)$; $z^e(S) = c^e(S)$

⟨statement index⟩ $a^i = w^i$; $b^i = x^i$; $s^i = x^i$; $f^i = y^i$; $c^i = z^i$

⟨label function⟩ $a^a = w^a$; $b^a = x^a$; $s^a = x^a$; $f^a = y^a$; $c^a = z^a$

⟨block index⟩ $a^j = w^j$; $b^j = x^j$; $s^j = x^j$; $f^j = y^j$; $c^j = z^j$

⟨block count⟩ $a^k = w^k$; $b^k = x^k$; $s^k = x^k$; $f^k = y^k$; $c^k = z^k$

⟨block determination⟩ $w^h = a^h$; $x^h = b^h$ or s^h; $y^h = f^h$; $z^h = c^h$

⟨number of statements⟩ w^n a^n; x^n b^n s^n; y^n f^n; z^n c^n

⟨enviroment⟩ $a^y = w^y$; $b^y = x^y$; $s^y = x^y$; $f^y = y^y$; $c^y = z^y$

⟨exit⟩ $w^x(S) = a^x(S)$; $x^x(S) =$ if $b^u(S)$ then $b^x(S)$ else if $b^m(S) \neq$ normal; then $b^m(S)$ else $s^x(b^1(S))$; $y^x(S) =$ if $i^v(S)$ then $f^x(i^s(S))$ else $i^s(S)$; $z^x(S) = c^x(S)$

⟨local label function⟩^{b}

$w^b = \emptyset$; $x^b = \emptyset$; $y^b = \emptyset$; $z^b = \{(a, z^i)\} \cup c^b$

⟨block synthesized index⟩ $a^s = w^s$; $b^s =$ if $x^h =$ block then $(l_1, \ldots l_n, 1)$ where $(l_1, l_2, \ldots l_n) = x^s$ else z^s; $s^s =$ if $x^h =$ block then $(l_1, \ldots l_n, 2)$ where $(l_1, \ldots l_n) = x^s$ else x^s; $f^s = y^s$; $c^s = z^s$

For Statements

⟨for list element⟩ x::=⟨arithmentic expression⟩c; y::=⟨arithmetic expression⟩ e 'step' ⟨arithmetic expression⟩ f 'until' ⟨arithmetic expression⟩ g, z::=⟨arithmetic expression⟩h while ⟨Boolean expression⟩b

⟨expanded form⟩^{f}

$$x^f = \begin{cases} L_k: & v := c \\ a_k: & s \end{cases}$$

$$y^f = \begin{cases} & v := e \\ L_k: & \text{if } [(v-g) \times \text{sign } (f) > 0] \text{ then go to } a_k; \\ & \quad s \\ & v := v + f; \\ & \text{go to } L_k; \\ a_k: & \quad ; \end{cases}$$

$$z^f = \begin{cases} L_k: & v := h; \\ & \text{if } \neg\, b \text{ then go to } a_k; \\ & \quad s \\ & \text{go to } L_k; \\ a_k: & \quad ; \end{cases}$$

for list x::=⟨for list element⟩ r; y::=⟨for list⟩ s ',' ⟨for list element⟩ t

⟨for list index⟩^{k}

$x^k = 1$

$y^k = s^k + 2$

⟨expanded label form⟩^{p}

$r^p = a$ with $L_k^{a=r^f}$ replaced by x^k and a_k replaced by $x^k + 1$

$t^p = b$ with $L_k^{b=t^f}$ replaced by y^k and a_k replaced by $y^k + 1$

this makes the labels in the expanded form well defined

⟨expanded form⟩^{f}

$x^f = r^p_j$ $y^f =$ follow (s^f, t^p)

where the function follow means the expanded code of t^p follows the expanded code of s^f

⟨for clause⟩ f::=⟨variable⟩ r ':=': '

⟨for list⟩ x 'do'

⟨expanded form⟩^{f}

$f^f =$ Y with the dummy variable v in x^f replaced by the control variable r

⟨for statement⟩ x::= ⟨for clause⟩ f ⟨statement⟩ t; y::=⟨label⟩ a ':'

⟨for statement⟩ g

⟨expanded form⟩^{f}

$x^f =$ begin Y with s replaced by t end $= f^f$

$y^f =$ a: g^f

It is redily seen that x^f and y^f are compound statements by construction. The effects and exits may then be evaluated by the machinery already developed previously.

⟨effect⟩ $x^e(S) = S'$ where $S' = x^{f^e}(S)$ with control variable undefined; $y^e(S) = S'$ where $S' = y^{f^e}(S)$ with the control variable undefined.

⟨exit⟩ $x^x(S) = x^{f^x}(S)$; $y^x(S) = y^{f^x}(S)$

⟨statement index⟩ $t^i = x^i$; $g^i = y^i$

⟨label function index⟩ $t^j = x^j$; $g^j = y^j$

⟨block count⟩ $t^k = x^k$; $g^k = y^k$

⟨block determination⟩ $x^h = t^h$; $y^h = g^h$

⟨number of statements⟩ $x^n = t^n$; $y^n = g^n$

⟨environment⟩ $f^y = x^y$; $t^y = x^y$; $g^y = y^y$

⟨local label function⟩^{b}

$x^b = \emptyset$; $y^b = \{(a, y^i)\} \cup g^b$

⟨block synthesized index⟩ $t^s = x^s$; $g^s = y^s$

Declarations

⟨declaration⟩ w::=⟨type declaration⟩ t; x::=⟨array declaration⟩ a;
y::=⟨switch declaration⟩ s; z::=⟨procedure declaration⟩ p;
⟨block index⟩ $t^j = w^j$; $a^j = x^j$; $s^j = y^j$
⟨local environment⟩^{z}
$w^z = t^z$; $x^z = a^z$; $y^z = s^z$; $z^z = p^z$
⟨variable set⟩ $w^o = t^o$; $x^o = a^o$; $y^o = s^o$
⟨environment⟩ $a^y = x^y$; $s^y = y^y$; $p^y = z^y$
⟨label function⟩ $s^a = y^a$

Type Declaration

⟨type list⟩ x::=⟨simple variable⟩ v; y::=⟨simple variable⟩ w ',' ⟨type list⟩ z
⟨type set⟩^{t}
$z^t = y^t$
⟨variable set⟩ $x^o = \{v\}$ $y^o = \{w\} \cup z^o$
⟨block index⟩^{j}
⟨local environment⟩^{z}
$x^z = \{((v, \text{type}), x^t)\} \cup \{((v, \text{var}), x^j)\}$
$y^z = \{((w, \text{type}), y^t)\} \cup \{((w, \text{var}), y^j)\} \cup z^z$

⟨type⟩ r::='real'; i::='integer'; b::='Boolean'
⟨type set⟩ $r^t = \text{real}$; $i^t = \text{integer}$; $b^t = \{\text{true}, \text{false}\}$
⟨type declaration⟩ d::=⟨type⟩ t ⟨type list⟩ x
⟨type set⟩^{t}
$x^t = t^t$
⟨variable set⟩ $d^o = x^o$
⟨block index⟩ $\psi^j = d^j$
⟨local environment⟩ $d^z = x^z$

Array Declarations

⟨lower bound⟩ x::=⟨arithmetic expression⟩ a

⟨value⟩ $x^v(S) = \underline{\text{if}}\ a^x(S) \neq \text{normal}\ \underline{\text{then}}\ \text{none}\ \underline{\text{else}}\ a^v(S)$

⟨exit⟩ $x^x(S) = a^x(S)$

⟨side effect⟩ $x^s(S) = a^s(S)$

⟨environment⟩ $a^y = x^y$

⟨upper bound⟩ y::=⟨arithmetic expression⟩ b

⟨value⟩ $y^v(S) = \underline{\text{if}}\ b^x(S) \neq \text{normal}\ \underline{\text{then}}\ \text{none}\ \underline{\text{else}}\ b^v(S)$

⟨exit⟩ $x^x(S) = a^x(S)$

⟨side effect⟩ $y^x(S) = b^x(S)$

⟨environment⟩ $b^y = y^y$

⟨bound pair⟩ x::=⟨lower bound⟩ a ':' ⟨upper bound⟩ b

$(a^v \leq b^v)$

⟨set of n- tuple⟩ $x^u = \{k: a^v \leq k \leq b^v\}$

⟨environment⟩ $a^y = x^y$; $b^y = x^y$

⟨bound pair list⟩ x::= ⟨bound pair⟩ a; y::=⟨bound pair list⟩ z ','
⟨bound pair⟩ b

⟨set of n-tuples⟩ $x^u = a^u$; $y^u = z^u \times b^u$

⟨rank count⟩ $x^n = 1$; $y^n = z^n + 1$

⟨environment⟩ $a^y = x^y$; $z^y = y^y$;; $b^y = y^y$

⟨array segment⟩ x::=⟨array identifier⟩ i ⟨subscript bound list⟩ b;
y::=⟨array identifier⟩ i ',' ⟨array segment⟩ z
⟨type set⟩^{t}
$z^t = y^t$

⟨set of n-tuples⟩ $x^u = b^u$; $y^u = z^u$

⟨rank count⟩^{n}

⟨rank⟩ $x^r = b^n$; $y^r = z^r$

⟨block index⟩^{j}

⟨local environment⟩ $x^z = \{(((i,k),\underset{\sim}{type}),x^t);\ k \in b^u\} \cup \{((i,\underset{\sim}{rank}),x^r)\}$
$\cup \{((i,\underset{\sim}{var}),x^j)\}$; $y^z = \{(((j,k),\underset{\sim}{type}),y^t):\ k \in z^u\}$
$\cup \{((j,\underset{\sim}{rank}),y^r)\} \cup \{((j,\underset{\sim}{var}),y^j)\} \cup z^z$

⟨environment⟩ $b^y = x^y$

⟨variable set⟩ $x^o = \{i\}$; $y^o = \{j\} \cup z^o$

⟨array list⟩ $\underset{\sim}{x}$::=⟨array segment⟩ $\underset{\sim}{s}$; $\underset{\sim}{y}$::=⟨array list⟩ $\underset{\sim}{z}$ ',' ⟨array segment⟩ $\underset{\sim}{t}$

⟨type set⟩^{t}

$s^t = x^t$; $z^t = y^t$; $t^t = y^t$

⟨local environment⟩ $x^z = s^z$; $y^z = z^z \cup t^z$

⟨block index⟩ $s^j = x^j$; $z^j = y^j$; $t^j = y^j$

⟨environment⟩ $s^y = x^y$; $z^y = y^y$; $t^y = y^y$

⟨variable set⟩ $x^o = s^o$; $y = z^o \cup t^o$

⟨array declaration⟩$\underset{\sim}{x}$::= '$\underset{\sim}{array}$' ⟨array list⟩ $\underset{\sim}{a}$; $\underset{\sim}{y}$::= ⟨type⟩ $\underset{\sim}{t}$ '$\underset{\sim}{array}$'
⟨array list⟩ $\underset{\sim}{b}$

⟨local environment⟩ $x^z = a^z$; $y^z = b^z$

⟨type set⟩^{t}

$a^t = \underset{\sim}{real}$; $b^t = t^t$

⟨environment⟩ $a^y = x^y$; $b^y = y^y$

⟨block index⟩ $a^j = x^j$; $b^j = y^j$

⟨variable set⟩ $x^o = a^o$; $y^o = b^o$

Switch Declararions

⟨Switch list⟩ x::=⟨designational expression⟩ a; y::=⟨switch list⟩
b ',' ⟨designational expression⟩ c
⟨environment⟩ $a^y = x^y$; $b^y = y^y$; $c^y = y^y$
⟨label function⟩ $a^a = x^a$; $b^a = y^a$; $c^a = y^a$
⟨range⟩ $x^n = 1$; $y^n = b^n + 1$
⟨pair set⟩ $x^p = \{(x^n, a)\}$
$y^p = \{(y^n, c)\} \cup b^p$

⟨switch identifier⟩ ::=⟨identifier⟩

⟨switch declaration⟩ x::= 'switch' ⟨switch identifier⟩ i ':=' ⟨switch list⟩ l
⟨label function⟩ $l^a = x^a$
⟨environment⟩ $l^y = x^y$
⟨range⟩^{n}
⟨pair set⟩^{p}
⟨local environment⟩ $x^z = \{((s, \text{switch range}), l^n)\} \cup$
$\{((s, k, \text{label}), m) : 1 \leq k \leq l^n \text{ and } (k, m) \in l^p\}$

Procedure Declarations

⟨formal parameter⟩ ::= ⟨identifier⟩

⟨formal parameter list⟩ x::= ⟨formal parameter⟩ p; y::= ⟨formal parameter list⟩ l ⟨parameter delimiter⟩ o ⟨formal parameter⟩ q

⟨parameter number⟩ $x^n = 1$

$y^n = l^n + 1$

⟨parameter variable set⟩^{s}

$x^s = \{p\}$; $y^s = \{q\} \cup l^s$

⟨parameter indexed set⟩^{x}

$x^x = \{(x^n, p)\}$; $y^x = \{(y^n, q)\} \cup l^x$

⟨formal parameter part⟩ x::= ⟨empty⟩ e; y::= '(' ⟨formal parameter list⟩ f ')'

⟨parameter number⟩ $x^n = o$; $y^n = f^n$

⟨parameter variable set⟩ $x^s = \emptyset$; $y^s = f^s$

⟨parameter index set⟩ $x^x = \emptyset$; $y^x = f^x$

⟨identifier list⟩ x::= ⟨identifier⟩ i; y::= ⟨identifier list⟩ l ',' ⟨identifier⟩ j

⟨ident number⟩ $x^k = 1$; $y^k = l^k + 1$

⟨ident set⟩ $x^p = \{i\}$; $y^p = \{j\} \cup l^p$

⟨value part⟩ x::= 'value' ⟨identifier list⟩ i; y::= ⟨empty⟩ e

⟨ident number⟩^{k}

⟨ident set⟩^{p}

⟨value number⟩ $x^l = i^k$

⟨value set⟩ $x^q = i^p$

⟨specifier⟩ p::='string'; q::=⟨type⟩ a; r::='array ; s:: ⟨type⟩ b 'array'; t::='label'; u::='switch'; v:: 'procedure'; w::=⟨type⟩ c 'procedure'

⟨parameter type⟩^{r} p^{r}= string q^{r}= a^{t}; r^{r}= array; s^{r}= concat (b^{t},array); t^{r}= label; u^{r}= switch ; v^{r}= procedure; w^{r}=concat (c^{t},procedure)

⟨type set⟩^{t}

⟨specification part⟩ x::=⟨empty⟩ e; y::=⟨specifier⟩ s ⟨identifier list⟩ l ';' ; z::=⟨specification part⟩ p ⟨specifier⟩ t ⟨identifier list⟩ m

⟨indent number⟩^{n}

⟨indent set⟩^{p}

⟨parameter type⟩^{r}

⟨specification set⟩ $x^{c}=\emptyset$; $y^{c}=\{x_1,\ldots x_n\}$ where each x_i is of the form (v_i,s^{r}) for $1\leq i\leq l^{n} \wedge v_i \bigcup l^{p}$.

$z^{c}=\{y_1,\ldots y_n\}\bigcup p^{c}$ where each y_i is of the form (v_i,t^{r}) for $1\leq i\leq m^{n} \wedge v_i \in m^{p}$

⟨procedure heading⟩ h::=⟨procedure identifier⟩ p ';' ⟨formal parameter pair⟩ f ';' ⟨value part⟩ v ⟨specification part⟩ s

⟨parameter number⟩ h^{n}= f^{n}

⟨parameter variable set⟩ h^{s}= f^{s}

⟨parameter index set⟩^{x}

⟨value number⟩^{l}

⟨value set⟩^{q}

⟨specification set⟩^{c}

⟨local environment⟩^{z}

$h^z = R \cup S \cup T \cup V$ where R,S,T and V are sets of the following form: S has elements of the form $((p, v_i,$ specification$), s^r)$ where each element of S of that form has a 1;1 correspondence with elements in s^c of the form (v^i, s^r)

$v = \{x_1, \ldots x_k, y_1 \ldots y_m\}$ where $k = v^i$ and $k + m = f^n$ each x_i is of the form $((v_i,$ pcm$),$ value$) \forall\ v_i \in (h^s \wedge v^q)$ each y_i is of the form $((v_i,$pcm$),$ name$)$ where $v_i \in h^s$ but $v_i \notin v^q$

T is a set whose elements are of the form:

$(((p,i),$par, index$),v_i) \forall\ v_i \in h^s$, p is the procedure name and i is the index number of the formal parameter v_i. There is a 1:1 correspondence between every $(i,v_i) \in f^x$ and $(((p,i),$par, index$)) \in T$.

R is a set of the form $\{((p,$ parameter number$), h^n), \ldots\}$

Procedure Statements and Function Designators

⟨actual parameter⟩ r::=⟨string⟩ a; s::=⟨expression⟩ b; t::=⟨array identifier⟩ c; u::=⟨switch identifier⟩ d; v::=⟨procedure identifier⟩ e

⟨actual parameter type⟩ r^d=string; s^d= expression ; t^d= array; u^d= switch; v^d= procedure

⟨procedure identifier⟩ ::=⟨identifier⟩

⟨letter string⟩ ::=⟨letter⟩ | ⟨letter string⟩ ⟨letter⟩

⟨parameter delimeter⟩ ::=',' | ')' ⟨letter string⟩ ':('

⟨actual parameter list⟩ x::=⟨actual parameter⟩ p; y::=⟨actual parameter list⟩ l ⟨parameter delimiter⟩ o ⟨actual parameter⟩ q

⟨actual parameter number⟩ x^k l

$y^k = l^k + 1$

⟨actual parameter set⟩^{a}

$x^a = \{p\}$; $y^a = \{q\} \cup l^a$

⟨actual parameter indexed set⟩^{b}

$x^b = \{(x^k, p)\}$; $y^b = \{(y^k, q)\} \cup l^b$;

⟨parameter type function⟩^{c}

$x^c = \{((x^k, p), p^d)\}$; $y^c = \{((y^k, q), q^d)\} \cup l^c$

⟨actual parameter type⟩^{d}

⟨actual parameter part⟩ x::= ⟨empty⟩ e; y::= '(' ⟨actual parameter list⟩ a ')'

⟨actual parameter number⟩ $x^k = \phi$; $y^k = a^k$

⟨actual parameter set⟩ $x^a = \emptyset$; $y^a = a^a$

⟨actual parameter index set⟩ $x^b = \emptyset$; $y^b = a^b$

⟨parameter type function⟩^{c} $x^c = \emptyset$; $y^c = a^c$

⟨procedure statement⟩ t::=⟨procedure identifier⟩ p ⟨actual parameter part⟩ a

⟨actual parameter number⟩^{k}

⟨actual parameter set⟩^{a}

⟨actual parameter index set⟩^{b}

⟨parameter type function type function⟩^{c}

⟨environment⟩ $a^y = t^y$; $p^y = t^y$

⟨initial effect⟩ $t^f(S) = S'$

Ther initial effect may first be seen by describing it in words. It is the effect of assigning the actual parameters to the formal parameters. The formal parameters are found by first finding out how many there are by t^y(p,parameter number)$= n$. Each of the formal parameters are found by evaluating $t^y((p,i),\underset{\sim}{par.\ index}) = v_i \; \forall i \ni 1 \leq i \leq n$. The formal parameters are then checked to see if they were called by name or by value. This can be done by evaluating $t^y(v_i,\underset{\sim}{parm})$. If they are called by value, the formal parameter v_i should be given the value of the actual parameter which is $t^b(i) = r_i$. Before this assignment may be made, however, the formal parameter name should be checked to see if they conflict with any variable names already in the domain of the state vector S. If there is a conflict, the conflicting names already in the domain of the state vector should be stacked with their current values. This procedure of stacking variables is much like the procedure done earlier with stacking variables

for entry into blocks. In order to avoid redundancy, I will omit the details of the stacking operations. During these assignments the specifications will serve as declarations for the formal parameters.. Finally the actual parameters have to be checked to see if they are congenial with their matching formal parameters. This is done by checking the specificaton of the formal parameter given by $t^y(p,v_i,\underline{specification})$ and the type of the actual parameter given by $t^a(i,r)$. If they are not congenial, as assigning an array to a variable, an error should be denoted. The effect of this section is essentially the effect of assigning these actual parameters to the formal parameters. If they are called by value the effect is exactly like the effect of having assignment statements at the beginning of the procedure. Like any assignment they may have improper exits in which case the the procedure will not be executed.

⟨effect⟩ $t^e(S) = \rho^e(S')$

⟨exit⟩ $t^x(S) = \rho^x(S')$

⟨function designator⟩ f::=⟨procedure identifier⟩ p ⟨actual parameter part⟩ a

The semantics for the function designator are the same as those of the procedure statement except the ⟨effect⟩ attribute is replaced by the ⟨side effect⟩ attribute.

C H A P T E R T H R E E

A D E S C R I P T I O N O F B A S I C

(as implemented on the Hewlett-Packard 2000B)

B Y

E U G E N E P O W E R S

PREFACE

The purpose of this paper is to describe scientifically the programming language BASIC using the techniques of W. D. Maurer as presented in Memorandum No. ERL-M368 of the College of Engineering, University of California at Berkeley.

The reference used for determining the syntax and semantics of the language was 2000B: A Guide to Time-shared BASIC provided by the Hewlett Packard company. The description given in the paper is complete except for two deletions from the original language. The potential for chaining programs and all matrix operations have been left out. Both could be added with a minimum change in the present description.

One assumption was made in regards to the semantics of the language. This assumption was that it should be illegal to jump to statements within the scope of a for loop. (from outside the loop)

Key words in the description are underlined. A full description of their meanings is given in the appendix. An alphabetical index to location of nonterminals is included at the end.

⟨evaluated subprogram⟩ a::=⟨subprogram⟩b

⟨files environment⟩^{q} $a^{q}=b^{q}$

⟨environment⟩^{z} $a^{z}=b^{z}$

⟨section length⟩^{k} $a^{k}=b^{k}$

⟨end index⟩^{N} $b^{N}=a^{N}$

⟨start index⟩^{s} $b^{s}=a^{s}$

⟨global label function⟩^{A} $b^{A}=a^{A}$

⟨global environment⟩^{Z} $b^{Z}=a^{Z}$

⟨generalized effect⟩^{f} $a^{f}((S,k))=$ if a^{s} k $a^{s}+a^{k}$ then $a^{f}(b^{p}((S,k)))$ else (S,k)

⟨effect⟩^{e} $a^{e}=$if $a^{f}((S,k))=(S',j)$ then S'

⟨exit index⟩^{x} $a^{x}=$if $a^{f}((S,k))=(S',j)$ then j

⟨BASIC program⟩ a::=⟨evaluated subprogram⟩b carriage return⟨seq no.⟩c ' END '

⟨start index⟩^{s} $b^{s}=1$

⟨end index⟩^{N} $b^{N}=b^{k}+1$

⟨global label function⟩^{A} $b^{A}=(c,b^{N})$

⟨effect⟩^{e} $a^{e}=b^{e}(S)$ where S is the starting state

⟨global environment⟩^{Z} $b^{Z}=$null

⟨subprogram⟩ a::=⟨seq no.⟩b⟨elementary statement⟩c ;
d::=⟨seq no.⟩e⟨declaration⟩f ;
g::=⟨seq no.⟩h⟨files statement⟩i ;
j::=⟨subprogram⟩k carriage return⟨seq no.⟩l⟨elementary statement⟩m ;
n::=⟨subprogram⟩o carriage return⟨seq no.⟩p⟨declaration⟩q;
r::=⟨subprogram⟩s carriage return⟨seq no.⟩t⟨files statement⟩u ;
v::=⟨seq no.⟩w⟨for section⟩x ;
y::=⟨subprogram⟩z carriage return⟨seq no.⟩A⟨for section⟩B

⟨files environment⟩^{q} $g^q(\underline{\text{defined files}})=i^z$; $r^q(\underline{\text{defined files}})=s^q(\underline{\text{defined files}})\ u^z$;

$v^q(\underline{\text{defined files}})=x^{sq}(\underline{\text{defined files}})$;

$y^q(\underline{\text{defined files}})=z^q(\underline{\text{defined files}})\ B^{sq}(\underline{\text{defined files}})$

(restriction:: the cardinality of the sets named on the left of the equality is less than 17)

(restriction:: $s^q(\underline{\text{defined files}}) \cap u^z = \underline{\text{null}}$ and $z^q(\underline{\text{defined files}}) \cap B^{sq}(\underline{\text{defined files}}) = \underline{\text{null}}$)

$m^q(\underline{\text{defined files}}) = k^q(\underline{\text{defined files}})$

⟨environment⟩^{z} $d^z=f^z$; $j^z=k^z$; $m^z=j^z$; $n^z=o^z \cup q^z$; $r^z=s^z$; $v^z=x^{sz}$;
$y^z=z^z \cup B^{sz}$;

(restriction:: let O be the set of first elements in o^z
let Q " q^z
let Z " z^z
let B' " B^{sz}
then $O \cap Q=\underline{\text{null}}$ and $Z \cap B'=\underline{\text{null}}$)

$c^z=d^z$; $x^z=\underline{\text{null}}$; $B^z=y^z \cup z^z$; $m^z=j^z \cup k^z$

⟨subprogram⟩ (cont'd)

⟨section length⟩^{k} $a^k=1$; $d^k=1$; $g^k=1$; $j^k=k^k+1$; $n^k=o^k+1$; $r^k=s^k+1$; $v^k=x^k$; $y^k=z^k+B^k$

⟨label function⟩^{a} $a^a=a^A\cup(b,a^s)$; $d^a=d^A\cup(e,d^s)$; $g^a=g^A\cup(h,g^s)$; $j^a=k^a\cup(l,m^i)$; $n^a=o^a\cup(p,q^i)$; $r^a=s^a\cup(t,u^i)$; $v^a=v^A\cup(w,v^s)$; $y^a=z^a\cup(A,B^i)$

⟨statement index⟩^{i} $a^i=a^s$; $d^i=d^s$; $g^i=g^s$; $v^i=v^s$; $m^i=j^s+k^k-1$; $q^i=n^s+o^k-1$ $u^i=r^s+s^k-1$; $B^i=y^s+z^k-1$

⟨start index⟩^{s} $o^s=n^s$; $k^s=j^s$; $s^s=r^s$; $z^s=y^s$

⟨end index⟩^{N} $c^N=a^N$; $m^N=j^N$; $x^N=v^N$; $B^N=y^N$

⟨single step function⟩^{p} $a^p((S,a^s))=(c^e(S),c^x(S))$; $d^p((S,d^s))=(S,d^s+1)$; $g^p((S,g^s))=(S,g^s+1)$; $j^p((S,K))=k((S,K))$ for j^s K j^s+k^k and $j^p((S,j^s+k^k))=(m^e(S),m^x(S))$; $n^p((S,K))=o^p((S,K))$ for n^s K n^s+o^k and $n^p((S,n^s+o^k))=(S,n^s+o^k+1)$; $r^p((S,K))=s^p((S,K))$ for r^s K r^s+s^k and $r^p((S,r^s+s^k))=(S,r^s+s^k+1)$; $v^p((S,v^s))=(x^e(S),x^x(S))$; $y^p((S,K))=z^p((S,K))$ for y^s K y^s+z^k and $y^p((S,y^s+z^k))=(B^e(S),\ B^x(S))$

⟨declaration⟩ a::=⟨def statement⟩b; c::=⟨dim statement⟩d

⟨environment⟩^{z} $a^{z}=b^{z}$; $c^{z}=d^{z}$

⟨elementary statement⟩ a::=⟨let statement⟩b; c::=⟨if statment⟩d;
e::=⟨goto statement⟩f; g::=⟨gosub statement⟩h; i::=⟨return statement⟩j;
k::=⟨stop statement⟩m; n::=⟨data statment⟩o; p::=⟨restore statement⟩q;
r::=⟨read statemnt⟩s; t::=⟨read file statment⟩u; v::=⟨input statement⟩w;
x::=⟨enter statement⟩y::=⟨null print statement⟩z;
A::=⟨print statement⟩B; C::=⟨file write statement⟩D

⟨label function⟩^{a} $d^{a}=c^{a}$; $f^{a}=e^{a}$; $h^{a}=g^{a}$

⟨environment⟩^{z} inherited intact in every case

⟨files environment⟩^{q} inherited intact in every case

⟨statement index⟩^{i} inherited intact in every case

⟨exit index⟩^{x} synthesized intact in every case

⟨effect⟩^{e} synthesized intact in every case

⟨end index⟩^{N} $m^{N}=k^{N}$

⟨seq no.⟩ a::=⟨integer⟩b

(restriction:: $1 \leq b^{v} \leq 9999$)

PASSIVE STATEMENTS

⟨def statement⟩ a::= ' DEF ' ⟨function identifier⟩b '('⟨simple variable⟩c ')''='⟨expression⟩d

⟨environment⟩^{z} $a^z = ((b,\underline{dummy}),c),c) \cup ((b,\underline{expression}),d)$

⟨dim statement⟩ a::= ' DIM ' ⟨dimspec⟩b; c::=⟨dim statement⟩d ',' ⟨dimspec⟩e

⟨environment⟩^{z} $a^z=b^z$; $c^z=d^z \cup e^z$

⟨dimspec⟩ a::=⟨array identifier⟩b '('⟨bound⟩c ')'

d::=⟨array identifier⟩e '('⟨bound⟩f ','⟨bound⟩g ')'

h::=⟨simple string variable⟩i '('⟨bound⟩j ')'

⟨environment⟩^{z} $a^z = ((b,\underline{rank}),1) \cup ((b,\underline{maxsize1}),c^v)$

$d^z = ((e,\underline{rank}),2) \cup ((e,\underline{maxsize1}),f^v) \cup ((e,\underline{maxsize2}),g^v)$

$h^z = ((i,\underline{maxsize}),j^v)$

(restriction:: $1 \leq c^v, f^v, g^v \leq 9999$ $1 \leq j^v \leq 72$)

⟨bound⟩ a::=⟨integer⟩ b

⟨value⟩^{v} $a^v=b^v$

⟨files statement⟩ a::=' FILES ' ⟨name⟩ b; c::=⟨files statement⟩^{i},'⟨name⟩ d

⟨exit index⟩^{x} $a^{x}=a^{i}+1$; $c^{x}=c^{i}+1$

⟨environment⟩^{z} $a^{z}=\{b\}$; $c^{z}=i^{z}\cup\{d\}$

(restriction:: $d\notin i^{z}$)

⟨effect⟩^{e} $a^{e}(S)=S'$ where $S'(\underline{FILEINDEX})=S(\underline{FILEINDEX})+1$

$S'(*)=1$ and $S'(\#\#(k))$, $S'(\$\$(k))$, and $S'(\&\&(k))$ are assigned values corresponding to an external file whose name is b, where

$*=$concat($\underline{FILEPTR}$,$S'(\underline{FILEINDEX})$)

$\#\#=$concat($\underline{FILETYPE}$, ")

$\$\$=$concat($\underline{FILELENGTH}$, ")

$\&\&=$concat($\underline{FILE}$, ")

(restriction:: $S'(FILEINDEX)\leq 16$)

⟨name⟩ a::=⟨literal string⟩ b

(restriction:: $b^{k}\leq 6$)

I/O STATEMENTS

⟨data statement⟩ a::= ' DATA ' ⟨constant⟩ b; c::=⟨data statement⟩d ',' ⟨constant⟩e

⟨effect⟩^{e} $a^e(S)=S'$ where $S'(\underline{EODATA})=S(\underline{EODATA})+1$ and $S'(\underline{DATA}(S(\underline{EODATA})))=b^v$ and $S'(\underline{DATATYPE}(S(\underline{EODATA})))=b^t$ and(if $b^t=$ string then $S'(\underline{DATALENGTH}(S(\underline{EODATA})))=b^k$ and $S'=S$ otherwise

$c^e=S'$ where S' is as above substituting e for b and $d^e(S)$ for S

⟨exit index⟩^{x} $a^x=a^i+1$; $c^x=c^i+1$

⟨restore statement⟩ a::= ' RESTORE '

⟨effect⟩^{e} $a^e(S)=S'$ where $S'(\underline{DATAPTR})=1$ and $S'=S$ otherwise

⟨exit index⟩^{x} $a^x=a^i+1$

⟨read statement⟩ a::= ' READ ' ⟨any variable⟩b; c::=⟨read statement⟩d ',' ⟨any variable⟩e

⟨effect⟩^{e} $a^e(S)=$if $S(\underline{DATAPTR})\leq S(\underline{EODATA})$ then S' where $S'(\underline{DATAPTR})=S(\underline{DATAPTR})+1$ and $S'(b)=S(\underline{DATA}(S(\underline{DATAPTR})))$ and(if $b^t=$string then $S'(\text{concat}(b,\underline{length}))=S(\underline{DATALENGTH}(S(\underline{DATAPTR})))$) and $S'=S$ otherwise else S

(restriction:: if $b^t=$string then $a^z(b,\underline{maxsize})\geq S'(\text{concat}(b,\underline{length}))$

(restriction:: $b^t=S(\underline{DATATYPE}(S(\underline{DATAPTR})))$)

$c^e(S)$ same as $a^e(S)$ substituting e for b and c for a and $a^e(S)$ for S

⟨read statement⟩ (continued)

⟨exit index⟩^{x} $a^x=a^i+1$; $c^x=c^i+1$

⟨any variable⟩ a::=⟨simple string variable⟩b; c::=⟨simple variable⟩d

⟨type⟩^{t} a^t=string; c^t=number

⟨part file read statement⟩ a::= ' READ ' ⟨file reference⟩b

⟨filename⟩^{f} $a^f=b^f$

⟨file reference⟩ a::= '#' expression b

⟨filename⟩^{f} a^f=rnd(b^v(S))

⟨read file statement⟩ a::=⟨part file read statement⟩b ';' ⟨any variable⟩c

d::=⟨read file statement⟩e ','⟨any variable⟩f

⟨filename⟩^{f} $a^f=b^f$

⟨exit index⟩^{x} $a^x=a^i+1$; $d^x=d^i+1$

⟨effect⟩^{e} a^e(S)=if S(##(**))≠ endoffilemark then S' where

S'(*)=S(*)+1 and S'(c) = S(##(**)) and (if c^t= string

then S'(concat(c,length))=S($$(**))) and S'=S

otherwise else S

(restriction:: if c^t=string then a^z(c,maxsize) ≥ S'(concat(c,length));

(restriction:: c^t=S(&&(**)))

see next page for defns of **, ##, and &&

⟨read file statement⟩ (continued)

⟨effect⟩ (continued)

where *= concat(FILEPTR,b^f) **=S(*)

##=concat(FILE,b^f) $$=concat(FILELENGTH,b^f)

&&=concat(FILETYPE,b^f)

(restriction:: $b^f \in a^q$(defined files))

d^e(S) same as a^e(S) substituting d for a and f for c and a^e(S) for S

⟨input statement⟩ a::= 'INPUT ' ⟨any variable⟩b; c::=⟨input statement⟩d ',' ⟨any variable⟩e

⟨exit index⟩ $^x a^x = a^i + 1$; $c^x = c^i + 1$

⟨effect⟩^{e} a^e(S) is as in a^e(S) for ⟨read file statement⟩ substituting b for c and letting ##=TTYFILE $$=TTYLENGTH

**=S(TTYPTR)

&&=TTYTYPE

similarly c^e(S) by substituting c for a and e for f and " for S

(note: disregard the restriction filename)

⟨enter statement⟩ a::=' ENTER ' '#'⟨variable⟩b

c::=' ENTER '⟨variable⟩d ','⟨variable⟩e ','⟨variable⟩f

g::= ' ENTER '⟨variable⟩h ','⟨variable⟩i ','
⟨simple string variable⟩j

k::='ENTER ','#'⟨variable⟩m ','⟨variable⟩n ','⟨variable⟩o
⟨any variable⟩p

⟨exit index⟩ $a^x=a^i+1$; $c^x=c^i+1$; $g^x=g^i+1$; $k^x=k^i+1$

⟨effect⟩ $a^e(S)=S'$ where $S'(b)=S(\underline{TTYNUMBER})$ and $S'=S$ otherwise

$c^e(S)=S'$ where if $S(\underline{TTY}(S(\underline{TTYPTR})))\neq\underline{endoffile}$ then

($S'(f)=$ if $S(\underline{TTYTIME}(S(\underline{TTYPTR})))\ d^v(S)$ then(if

$S(\underline{TTYTYPE}(S(\underline{TTYPTR})))=\underline{number}$ then $S(\underline{TTY}($

$(\underline{TTYPTR}))$ else $S(f)$) else $S(f)$

$S'(e)=$ if $S(\underline{TTYTIME}(S(\underline{TTYPTR})))\ d^v(S)$ then (if
$S(\underline{TTYTYPE}(S(\underline{TTYPTR})))=\underline{number}$ then
$S(\underline{TTYTIME}(S(\underline{TTYPTR})))$ else $-S(\underline{TTYTIME}(S(\underline{TTYPTR})))$

else -256 and $S'(\underline{TTYPTR})=S(\underline{TTYPTR})+1$) else
$S'(e)=-256$

(restriction:: $d^v(S)\geq 1$ and $d^v(S)\leq 255$)

and $S'=S$ otherwise

$g^e(S)=S'$ where S' is as for $c^e(S)$ substituting

h for d, i for e, j for f, and string for number

with the following additions within the first then clause

$S'(\text{concat}(j,\underline{length}))=S(\underline{TTYLENGTH}(S(\underline{TTYPTR})))$

(restriction:: $S'(\text{concat}(j,\underline{length}))\leq c^z(j,\underline{maxsize})$)

$k^e(S)=S'$ where S' is determined as follows

if $p^t=\underline{number}$ then S' is as in $c^e(S)$ substituting

n for d, o for e, and p for f with addition below.

if $p^t=\underline{string}$ then S' is as in $g^e(S)$ substituting

n for h, o for i, and p for j with addition below.

In both cases the following addition is made to S'

$S'(m)=S(\underline{TTYNUMBER})$

⟨null print statement⟩ a::= ' PRINT '

⟨exit index⟩^{x} $a^{x}=a^{1}+1$

⟨effect⟩^{e} $a^{e}(S)=S'$, where if $S(\underline{PRINTER})=(x_1,\ldots,x_n)$ then $S'(\underline{PRINTER})=(\underline{emptyline},x_1,\ldots,x_n)$ and $S'(z)=S(z)$ for $z\neq\underline{PRINTER}$

⟨print statement⟩ a::=⟨null print statement⟩ b ⟨every variable⟩c

d::=⟨print statement⟩e ',' ⟨every variable⟩f

⟨exit index⟩ a =a + 1; d =d + 1

⟨effect⟩^{e} $a^{e}(S)=S'$, where if $S(\underline{PRINTER})=(x_1,\ldots,x_n)$ then $S'(\underline{PRINTER})=(c^{v}(S),x_1,\ldots,x_n)$ and $S'(z)=S(z)$ for $z\neq\underline{PRINTER}$

$d^{e}(S)=S''$, where if $S'=e^{e}(S)$ and $S'(\underline{PRINTER})=(x_1,\ldots,x_n)$ then $S''(\underline{PRINTER})=(x_1',\ldots,x_n)$ where $x_1'=\text{concat}(x_1,'\quad',f^{v}(S))$ and $S''(z)=S(z)$ for $z\neq\underline{PRINTER}$

⟨every variable⟩ a::=⟨variable⟩b ; c::=⟨string variable⟩d

⟨value⟩^{v} $a^{v}=b^{v}$; $c^{v}=\underline{convert}(d^{v}(S))$

⟨file write statement⟩ a::= ' PRINT '⟨file reference⟩b ','⟨write ex⟩c

d::=⟨file write statement⟩e ','⟨write ex⟩f

⟨exit index⟩^{x} $a^{x}=a^{i}+1$; $d^{x}=d^{i}+1$

⟨effect⟩^{e} $a^{e}(S)=S'$ where S' is defined as follows

if $c^{m}=$*true* then $S'(*)=S(*)+1$

$S'(\#\#(**))=c^{v}(S)$

$S'(\&\&(**))=c^{t}$

and if $c^{t}=$*string* then

$S'(\%\%(**))=c^{n}$

else $S'(\#\#(**))=$*endoffile*

at all other points in the domain $S'=S$, in both cases

where $*=$concat(*FILEPTR*,b^{f})

$**=S(*)$

$\#\#=$concat(*FILE*,b^{f})

$\&\&=$concat(*FILETYPE*,b^{f})

$\%\%=$concat(*FILELENGTH*,b^{f})

(restriction:: $b^{f}\in a^{z}$(*defined files*))

$d^{e}(S)=S'$ where S' is defined as in $a^{e}(S)$ but substituting $e^{e}(S)$ for S and f for c and d for a

⟨environment⟩^{z} $c^{z}=a^{z}$; $f^{z}=d^{z}$

⟨write exp⟩ a::=⟨expression⟩b; c::= 'END'; d::=⟨source string⟩e

⟨type⟩^{t} $a^{t}=$*number* ; $d^{t}=$*string*

⟨clue⟩^{m} $a^{m}=d^{m}=$*true*; $c^{m}=$*false*

⟨value⟩^{v} $a^{v}(S)=b^{v}(S)$; $d^{v}(S)=e^{v}(S)$

⟨length⟩^{n} $d^{n}=e^{n}$

⟨environment⟩^{z} $b^{z}=a^{z}$; $c^{z}=d^{z}$

ACTIVE NON I/O STATEMENTS

⟨let statement⟩ a::= ' LET '⟨leftpart⟩b⟨expression1⟩c;
d::= ' LET '⟨string variable⟩e '='⟨source string⟩f;
k::=⟨string variable⟩g '=' ⟨source string⟩h;
m::=⟨left part⟩i '=' ⟨expression1⟩j

⟨environment⟩^{z} $b^z=a^z$; $c^z=a^z$; $e^z=d^z$; $f^z=d^z$; $h^z=k^z$; $g^z=k^z$; $i^z=m^z$; $j^z=m^z$

⟨effect⟩^{e} $a^e(S)=S'$ where $S'(z)=S(z)$ for $z\notin b^L(S)$

and $S'(z)=c^v(S)$ for $z\in b^L(S)$;

$m^e(S)=S'$ where $S'(z)=S(z)$ for $z\notin i^L(S)$

and $S'(z)=j^v(S)$ for $z\in i^L(S)$;

$d^e(S)=S'$ where $S'(e)=f^v(S)$ and $S'(\text{concat}(e,\underline{\text{length}}))=f^n$

and $S'(z)=S(z)$ otherwise;

$k^e(S)=S'$ where $S'(g)=h^v(s)$ and $S'(\text{concat}(e,\underline{\text{length}}))=h^n$

and $S'(z)=S(z)$ otherwise

⟨length⟩^{n} (restriction:: $d^z(e,\underline{\text{maxsize}})\geq f^n$)

(" $k^z(g,\underline{\text{maxsize}})\geq h^n$)

⟨statement index⟩^{i}

⟨exit index⟩^{x} $a^x=a^i+1$; $k^x=k^i+1$; $d^x=d^i+1$; $m^x=m^i+1$

⟨source string⟩ a::=⟨string variable⟩b; c::=⟨literal string⟩d

⟨length⟩^{n} $a^n=S(\text{concat}(b,\underline{\text{length}}))$; $c^n=d^k$

⟨expression1⟩ a::=⟨conjunction1⟩ b; c::=⟨expression1⟩d ' OR ' ⟨conjunction1⟩e

semantics are identical to those of⟨expression⟩

⟨conjunction1⟩ a::=⟨relation1⟩b; c::=⟨conjunction1⟩d ' AND '⟨relation⟩e

semantics are identical to those of⟨conjunction⟩

⟨relation1⟩ a::=⟨minmax⟩b ; c::= '('⟨minmax⟩d⟨relational op⟩e⟨minmax⟩f ')'

semantics identical to those for⟨relation⟩

⟨if statement⟩ a::= ' IF ' ⟨decision expression⟩b ' THEN ' ⟨sequence number⟩c

d::= ' IF END# ' ⟨file formula⟩e 'THEN '⟨sequence number⟩f

⟨environment⟩^{z} $b^z=a^z$; $e^z=d^z$;

⟨label function⟩^{a}

⟨statement index⟩^{i}

⟨exit index⟩^{x} $a^x=$ if $b^v\neq 0$ then a^i+1 else $a^a(c)$

$d^x=$ if $s(\text{concat}(\text{FILE},e^f)\ (s(\text{concat}(\text{FILEPTR},e^f))))=$ endoffilemark then $d^a(f)$ else d^i+1

⟨effect⟩^{e} $a^e(S)=S$; $d^e(S)=S$

⟨decision expression⟩ a::=⟨expression⟩b; c::=⟨string variable⟩d ⟨relational operator⟩e ⟨source string⟩f

⟨value⟩^{v} $a^v=b^v$; $c^v=d^{F(d^v,f^v)}$

⟨goto statement⟩ a::= ' GOTO '⟨sequence number⟩b; c::= ' GOTO ' ⟨expression⟩d ' OF '⟨sequence list⟩e

⟨effect⟩^{e} $a^e(S)=S$; $c^e(S)=S$

⟨label function⟩^{a} $b^a=a^a(b)$; $e^a=c^a$

⟨statement index⟩^{i}

⟨exit index⟩^{x} $a^x(S)=b^a$; $c^x(S)=$ if $e^{G(\text{rnd}(d^v(S)))}$ is defined then $e^{G(\text{rnd}(d^{\ }(S)))}$ else c^i+1

⟨sequence list⟩ a::=⟨sequence number⟩b; c::=⟨sequence list⟩d ',' ⟨sequence number⟩e

⟨sequence function⟩^{G} $a^G=(1,b^a)$; $c^G=d^G \cup (c^c,e^a)$

⟨list length⟩^{c} $a^c=1$; $c^c=d^c+1$

⟨label function⟩^{a} $b^a=a^a(b)$; $e^a=c^a(e)$

⟨gosub statement⟩ a::=' GOSUB ' ⟨sequence number⟩ b; c::=' GOSUB '
⟨expression⟩e 'OF'⟨sequence list⟩d

⟨exit index⟩^{x} $a^x=a^a(b)$; $c^x(S)=$ *if* $d^{G(\underline{rnd}(e^v(S)))}$ *is defined* *then*
" *else* c^i+1

⟨statement index⟩^{i}

⟨label function⟩^{a} $d^a=c^a$

⟨effect⟩^{e} $a^e(S)=S'$ where $S'(\underline{retstack}) = S(\underline{retstack})$ *PUSH* a^i+1

⟨return statement⟩ a::= ' RETURN '

⟨effect⟩^{e} a^e = S' where
$S(\underline{retstack})= S(\underline{retstack})$ *POP* . X; S'=S otherwise

⟨exit index⟩^{x} $a^x=X$

⟨stop statement⟩ a::= ' STOP'

⟨effect⟩^{e} $a^e(S)=S$

⟨exit index⟩^{x} $a^x=a^N$

⟨for statement⟩ a::= ' FOR ' ⟨simple variable⟩b '=' ⟨expression⟩c
' TO ' ⟨expression⟩d carriage return
e::= ' FOR ' ⟨simple variable⟩f '=' ⟨expression⟩g ' TO '
⟨expression⟩h ' STEP ' ⟨expression⟩i carriage return

⟨environment⟩^{z} $b^z=a^z$; $c^z=a^z$; $d^z=a^z$; $f^z=e^z$; $g^z=e^z$; $h^z=e^z$; $i^z=e^z$

⟨effect⟩^{e} $a^e(S)=S'$ where $S'(b)=c^v(S)$ $S'(\text{concat}(b,\underline{\text{limit}}))=d^v(S)$

$S'(\text{concat}(b,\underline{\text{step}}))=1$ $S'(\text{concat}(b,\underline{\text{loop}})=a^i+1$

and $S'(z)=S(z)$ otherwise

$b^e(S)=S'$ where $S'(f)=g^v(S)$ $S'(\text{concat}(f,\underline{\text{limit}}))=h^v(S)$

$S'(\text{concat}(f,\underline{\text{step}}))=i^v(S)$ $S'(\text{concat}(f,\underline{\text{loop}})=b^i+1$

and $S'(z)=S(z)$ otherwise

⟨exit index⟩^{x} $a^x=a^x+1$; $e^x=e^x+1$

⟨statement index⟩^{i}

⟨for variable⟩^{m} $a^m=b$; $e^m=f$

⟨for section⟩ a::=⟨for statement⟩b⟨subsection⟩c

⟨statement index⟩^{i} $b^i=a^i$; $c^i=a^i+1$

⟨environment⟩^{z} $b^z=a^z$; $c^z=a^z$

⟨label function⟩^{a} $c^a=a^a$

⟨for variable⟩^{f} (restriction:: $c^f=b^f$)

⟨effect⟩^{e} $a^e=c^e(b^e(S))$

⟨exit index⟩^{x} $a^x=c^x(b^e(S))$

⟨section length⟩^{k} $a^k=c^k+1$

⟨end index⟩^{N} $c^N=a^N$

⟨synth envir⟩^{sz} $a^{sz}=c^{sz}$

⟨synth files⟩^{sq} $a^{sq}=c^{sq}$

⟨subsection⟩c a::=⟨evaluated subprogram⟩b carriage return ⟨next statement⟩ c

⟨for variable⟩^{f} $a^{f}=c^{f}$

⟨start index⟩^{s} $b^{s}=a^{i}$

⟨global label function⟩^{A} $b^{A}=a^{a} \cup c^{a}$

⟨global environment⟩^{Z} $b^{Z}=a^{z}$

⟨effect⟩^{e} $a^{e}(S)=$ if $b^{x}(S)=c^{i}$ then (if $S'(c^{f},\text{step}) \times (S'(c^{f},\text{limit})-S'(c^{f})) < 0$ where $S'=c^{e}(b^{e}(S))$ then $a^{e}(S')$ else S') else $b^{e}(S)$.

⟨exit index⟩^{x} $a^{x}(S)=$if $b^{x}(S)=c^{i}$ then (if $S'(c^{f},\text{step}) \times (S'(c^{f},\text{limit})-S'(c^{f}))<0$ where $S'=c^{e}(b^{e}(S))$ then $a^{x}(S')$ else $c^{i}+1$) else $b^{x}(S)$.

⟨section length⟩^{k} $a^{k}=b^{k}+1$

⟨synth envir⟩^{sz} $a^{sz}=b^{z}$

⟨synth files⟩^{sq} $a^{sq}=b^{q}$(defined files)

⟨end index⟩^{N} $b^{N}=a^{N}$

⟨statement index⟩^{i} $c^{i}=b^{s}+a^{k}$

⟨next statement⟩ a::=⟨seq no.⟩b ' NEXT '⟨simple variable⟩c

⟨for variable⟩ a =c

⟨effect⟩^{e} $a^{e}(S)=S'$ where $S'(c)=S(c)+S(c,\text{step})$ and $S'=S$ otherwise

⟨label fn⟩^{a} $a^{a}=(b,a^{i})$

VARIABLES

⟨variable⟩ a::=⟨simple variable⟩x; b::=⟨subscripted variable⟩y;

⟨type⟩^{t} a^{t}=number; b^{t}=number;

⟨environment⟩^{z} $y^{z}=b^{z}$; (restriction:: $a^{z}(x,\underline{rank})=\underline{undefined}$)

⟨L-value⟩^{L} $a^{L}=x^{v}$; $b^{L}=y^{L}$

⟨simple variable⟩::=⟨letter⟩ / ⟨letter⟩⟨digit⟩

⟨letter⟩ a::='A'; b::='B'; c::='C'; d::='D'; e::='E'; f::='F';
g::='G'; h::='H'; i::='I'; j::='J'; k::='K'; l::='L';
m::='M'; n::='N'; o::='O'; p::='P'; q::='Q'; r::='R';
s::='S'; t::='T'; u::='U'; v::='V'; w::='W'; x::='X';
y::='Y'; z::='Z'

⟨value⟩^{v} a^{v}=33; b^{v}=34; c^{v}= 35; d^{v}= 36; e^{v}= 37; f^{v}=38; g^{v}=39; h^{v}= 40;
i^{v}= 41; j^{v}= 42; k^{v}=43; l^{v}=44; m^{v}=45; n^{v}=46; o^{v}=47; p^{v}=48;
q^{v}=49; r^{v}=50; s^{v}=51; t^{v}=52; u^{v}=53; v^{v}=54; w^{v}=55; x^{v}= 56;
y^{v}=57; z^{v}=58

⟨subscripted variable⟩ a::=⟨letter⟩b⟨sublist⟩c

⟨rank⟩^{r} $a^{r}=c^{n}$

⟨value⟩^{y} $a^{v}(S)=S(b(c^{i}(S))$

⟨L-value⟩^{L} $y^{L}=b(c^{i}(S))$

⟨environment⟩^{z} (restriction:: $a^{z}(b,\underline{rank})=a^{r}$)

(restriction:: $1 \leq c^{p}(1) \leq a^{z}(b,\underline{maxsize1})$ and

if $a^{r}=2$ then $1 \leq c^{p}(2) \leq a^{z}(b,\underline{maxsize2})$)

⟨sublist⟩ a::=⟨expression⟩b; c::=⟨expression⟩d ','⟨expression⟩e

⟨length⟩^{n} b^n=1; c^n=2;

⟨index value⟩^{i} $a^i(S)$=rnd($b^v(S)$); $c^i(S)$=concat(rnd($d^v(S)$), ',',rnd($e^v(S)$))

⟨sublist part⟩ $c^p(1)$=rnd($d^v(S)$); $c^p(2)$=rnd($e^v(S)$)

⟨string variable⟩ a::=⟨simple string variable⟩b;
c::=⟨simple string variable⟩d⟨sublist⟩e

⟨value⟩^{v} a^v=S(b); c^v=if e^n=1 then break(e^i,S(d)) else save($e^p(2)$,break($e^p(1)$,S(d)))

⟨length⟩^{n} a^n=b^n; c^n=if e^n=1 then (if e^i<d^n then d^n+1-e^i else 0) else (if $e^p(2)$>$e^p(1)$ then (if $e^p(1)$>d^n then 0 else (if $e^p(2)$>d^n then d^n-$e^p(1)$+1 else $e^p(2)$-$e^p(1)$+1)) else 0.
(restriction::=$c^n \leq c^z$(c,maxsize))

⟨type⟩^{t} a^t=string; c^t=string

⟨simple string variable⟩ a::=⟨letter⟩b '$'

⟨length⟩^{n} a^n=S(concat(b,length))

EXPRESSIONS

⟨expression⟩a::=⟨conjunction⟩b; c::=⟨expression⟩d 'OR'⟨conjunction⟩e

⟨value⟩^{V} $a^V=b^V$; $c^V=$ if $d^V=0$ and $e^V=0$ then 0 else 1

⟨environment⟩^{Z} $b^Z=a^Z$; $d^Z=c^Z$; $e^Z=c^Z$

⟨conjunction⟩a::=⟨relation⟩b; c::=⟨conjunction⟩d 'AND'⟨relation⟩e

⟨value⟩^{V} $a^V=b^V$; $c^V=$ if $d^V\neq0$ then 1 else 0

⟨environment⟩^{Z} $b^Z=a^Z$; $d^Z=c^Z$; $e^Z=c^Z$

⟨relation⟩ a::=⟨minmax⟩b; c::=⟨minmax⟩d ⟨relational operator⟩e ⟨minmax⟩f

⟨value⟩^{V} $a^V=b^V$; $c^V=e^{F(d^V,f^V)}$

⟨environment⟩^{Z} $b^Z=a^Z$; $d^Z=c^Z$; $e^Z=c^Z$; $g^Z=f^Z$; $h^Z=f^Z$

⟨minmax⟩a::=⟨sum⟩b; c::=⟨minmax⟩d 'MIN'⟨sum⟩e; f::=⟨minmax⟩g 'MAX'⟨sum⟩h

⟨value⟩^{V} $a^V=b^V$; $c^V=$ if $d^V<e^V$ then d^V else e^V; $f^V=$ if $g^V>h^V$ then g^V else h^V

⟨environment⟩^{Z} $b^Z=a^Z$; $d^Z=c^Z$; $e^Z=c^Z$; $g^Z=f^Z$; $h^Z=f^Z$

⟨sum⟩ a::=⟨term⟩b; c::=⟨sum⟩d '+' ⟨term⟩e; f::=⟨sum⟩g '-' ⟨term⟩h

⟨value⟩^{V} $a^V=b^V$; $c^V=d^V+e^V$; $f^V=g^V-h^V$

⟨environment⟩^{Z} $b^Z=a^Z$; $d^Z=c^Z$; $e^Z=c^Z$; $g^Z=f^Z$; $h^Z=f^Z$

⟨term⟩ a::=⟨subterm⟩b; c::=⟨term⟩d '*'⟨subterm⟩e; f::=⟨term⟩g '/' ⟨subterm⟩h

⟨value⟩^{V} $a^V=b^V$; $c^V=d^V\times e^V$; $f^V=g^V/h^V$

⟨environment⟩^{Z} $b^Z=a^Z$; $d^Z=c^Z$; $e^Z=c^Z$; $g^Z=f^Z$; $h^Z=f^Z$

⟨subterm⟩ a::=⟨denial⟩b; c::=⟨signed factor⟩d

⟨value⟩^{V} $a^V=b^V$; $c^V=d^V$

⟨environment⟩^{Z} $b^Z=a^Z$; $d^Z=c^Z$

⟨denial⟩a::=⟨factor⟩b; c::= 'NOT'⟨factor⟩d

⟨value⟩^{V} $a^V=b^V$; $c^V=$ if $d^V=0$ then 1 else 0

⟨environment⟩^{Z} $b^Z=a^Z$; $d^Z=c^Z$

⟨signed factor⟩ a::= '+' ⟨factor⟩g; c::= '-'⟨factor⟩d.

⟨value⟩^{V} $a^{V}=b^{V}$; $c^{V}=-d^{V}$

⟨environment⟩^{Z} $b^{Z}=a^{Z}$; $d^{Z}=c^{Z}$

⟨factor⟩ a::=⟨primary⟩b; c::=⟨factor⟩d '↑' ⟨primary⟩e

⟨value⟩^{V} $a^{V}=b^{V}$; $c^{V}=\exp(d^{V},e^{V})$

⟨environment⟩^{Z} $b^{Z}=a^{Z}$; $d^{Z}=c^{Z}$; $e^{Z}=c^{Z}$

⟨primary⟩a::=⟨variable⟩v; b::=⟨number⟩n; c::= '(' ⟨expression⟩e')';
d::= functional f

⟨value⟩^{V} $a^{V}=S(v)$; $b^{V}=n^{V}$; $c^{V}=e^{V}$; $d^{V}=f^{V}$

⟨environment⟩^{Z} $v^{Z}=a^{Z}$; $e^{Z}=c^{Z}$; $f^{Z}=d^{Z}$

⟨relational operator⟩a::='<'; b::='<='; c::='='; d::='#'; e::='<>';
f::='>='; g::='>'

⟨associated valuation⟩$^{F(n,p)}$ where n,p are ⟨numbers⟩

$a^{F(n,p)}$=<u>if</u> n<p <u>then</u> 1 <u>else</u> 0; $b^{F(n,p)}$=<u>if</u> n≤p <u>then</u> 1 <u>else</u> 0;

$c^{F(n,p)}$=<u>if</u> n=p <u>then</u> 1 <u>else</u> 0; $d^{F(n,p)}$=<u>if</u> n≠p <u>then</u> 1 <u>else</u> 0;

$e^{F(n,p)}$=<u>if</u> n≠p <u>then</u> 1 <u>else</u> 0; $f^{F(n,p)}$=<u>if</u> n≥p <u>then</u> 1 <u>else</u> 0

$g^{F(n,p)}$=<u>if</u> n>p <u>then</u> 1 <u>else</u> 0

⟨functional⟩ a::=⟨function identifier⟩b '(' ⟨expression⟩c ')';
d::=⟨predefined function⟩ e '(' ⟨expression⟩f ')'
g::= 'LEN' '(' ⟨string variable⟩h ')'

⟨value⟩^{V} $d^{V}=e^{G(f^{V})}$; $g^{V}=h^{n}$;

a^{V}= (<u>subst</u>(c^{V},a^{Z}(b,<u>dummy</u>),a^{Z}(b,<u>expression</u>)))^{V}

⟨function identifier⟩ ::= 'FN' ⟨letter⟩

⟨predefined function⟩ a::='SIN'; b::='COS'; c::='TAN'; d::='EXP';
e::='LOG'; f::='ABS; g::='SQR'; h::='INT'; i::='RND'; j::='SGN';
k::='TYP'; m::='TIM'

⟨functional valuation⟩$^{G(x)}$ where x is a ⟨number⟩

$a^{G(x)}$=sin(x) where x is taken to be in radians

<predefined function> ---Con't

$b^{G(x)}$ =cos(x) where x is taken to be in radians

$c^{G(x)}$ =tan(x) where x is taken to be in radians

$d^{G(x)}$ =exp(e,x)

$e^{G(x)}$ =ln(x) for x>0

$f^{G(x)}$ =|x| where || represents the absolute value function

$g^{G(x)}$ =$\sqrt{x}$ where $\sqrt{\ }$ represents the square root function

$h^{G(x)}$ =⌊x⌋ where ⌊ ⌋ represents the greatest integer function

$i^{G(x)}$ =n where $0 \leq n, m \leq 1$ and $p(i^{G(x)}=n) = p(i^{G(x)}=m)$ where p is probability

$j^{G(x)}$ =if x>0 then 1 else if x=0 then 0 else -1

$m^{G(x)}$ =if x=0 then current minute of the hour else if x=1 then current hour of the day else if x=2 then current day of the year else if x=3 then current year of the century

where 0 minutes 59
0 hours 23
1 day 366
0 year 99

CONSTANTS

⟨constant⟩c::=⟨number⟩x; d::= '-'⟨number⟩y; e::= '+'⟨number⟩z;
f::=⟨literal string⟩s

⟨value⟩^{V} $c^V=x^V$; $d^V=-y^V$; $e^V=z^V$; $f^V=s^V$ ---⟨length⟩^{k} $f^k=s^k$

⟨type⟩^{t} $c^t=$real; $d^t=$real; $e^t=$real; $f^t=$string

⟨number⟩n::=⟨decimal number⟩x; m::=⟨decimal number⟩y⟨exponent part⟩z

⟨value⟩^{V} $n^V=x^V$; $m^V=y^V$ x exp(10,z^V)

⟨decimal number⟩a::=⟨integer⟩x; b::=⟨integer⟩y '.' ; c::=⟨integer⟩u '.'
⟨integer⟩v; d::= '.' ⟨integer⟩w

⟨value⟩^{V} $a^V=x^V$; $b^V=y^V$; $c^V=u^V+v^V$x exp(10,$-v^k$); d =exp(10,$-w^k$)x w^V

⟨integer⟩a::=⟨digit⟩x; b::=⟨integer⟩y⟨digit⟩z

⟨value⟩^{V} $a^V=x^V$; $b^V=y^V$x 10 + z^V

⟨length⟩^{k} $a^k=1$; $b^k=y^k+1$

⟨digit⟩ a::='0'; b::='1'; c::='2'; d::='3'; e::='4'; f::='5'; g::='6'
h::='7'; i::='8'; j::='9'

⟨value⟩^{V} $a^V=0$; $b^V=1$; $c^V=2$; $d^V=3$; $e^V=4$; $f^V=5$; $g^V=6$; $h^V=7$; $i^V=8$;
$j^V=9$

⟨exponent part⟩e::= 'E'⟨integer⟩a; f::='E+'⟨integer⟩b; g::='E-'
⟨integer⟩c

⟨value⟩^{V} $e^V=a^V$; $f^V=b^V$; $g^V=-c^V$

⟨length⟩^{k} $e^k=a^k$; $f^k=b^k$; $g^k=c^k$ ($\leq e^k, g^k, f^k \leq 2$)

⟨literal string⟩ s::= ' '' '⟨character string⟩c' '' '

⟨value⟩^{V} $s^V=c^V$ ---⟨length⟩^{k} $s^k=c^k$.

⟨character string⟩c::=⟨character⟩d; e::=⟨character string⟩f⟨character⟩g

⟨value⟩^{V} $c^V=d^V$; $e^V=100$ x f^V+g^V

⟨length⟩^{k} $c^k=1$; $e^k=f^k+1$

⟨character⟩ a::='!'; b::='#'; c::='$'; d::='%'; e::='&'; f::='''';
g::='('; h::=')'; i::='*'; j::='+'; k::=','; k::='-';
l::='.'; m::='/'; n::=⟨digit⟩o; p::=':'; q::=';'; r::='<'
s::='='; t::='>'; u::='?'; v::='@'; w::=⟨letter⟩x; y::='['
z::='\'; A::=']'; B::='↑'; C::=' '

⟨value⟩ $a^V=1$; $b^V=2$; $c^V=3$; $d^V=4$; $e^V=5$; $f^V=6$; $g^V=7$; $h^V=8$; $i^V=9$;
$j^V=10$; $k^V=11$; $l^V=12$; $m^V=13$; $n^V=o^V + 14$; $p^V= 24$; $q^V= 25$;
$r^V=26$; $s^V=27$; $t^V=28$; $u^V=29$; $v^V=30$; $w^V=x^V$; $y^V=60$;
$z^V=61$; $A^V=62$; $B^V=63$; $C^V=0$

APPENDIX

null represents the null set

carriage return represents the carriage return character

if...then...else has its standard meaning (as in ALGOL)

concat(...,...) performs a standard concatenation of arguments

rnd(...) returns the rounded value of its argument (integer)

convert(...) converts a string back from its numeric value to its charcter form

a POP b pops an element from stack a into variable b

a PUSH b pushes b onto stack a

retstack a special variable to keep track of subroutine return points

subst(a,b,c) creates a new expression by substituting the actual variable a for the original variable (dummy) b in the expression c

File related key words:

The DATA file, the TTY, and external user files are represented by three parallel arrays with a common pointer denoting present location within the file. The 3 arrays contain the value of a record (one variable), its type, and if it is a string the third contains its length.

All other key words are used to form unique variable names and have straight forward interpertations.

C H A P T E R F O U R

A D E S C R I P T I O N O F P A L I I I

(PDP-8 assembly language)

B Y

R O N A L D O . N I C H O L S

INTRODUCTION

This syntactic and semantic description is for the Digital Equipment Corporation PDP-8/E PAL III assembler. The PDP-8/E is designed as a general purpose computer. It's basic processor is a single-address, fixed word length, parallel transfer unit using 12-bit, 2's complement arithmetic. Standard features include indirect addressing and facilities for instruction skip.

The PAL III symbolic assembler(PAL stands for Program Assembly Language) translates symbolic programs, which are written in the PAL III language, into binary-coded programs. It features symbolic refrences, origins, and expressions.

Five 12-bit registers are used to control computer operations, address memory, operate on data and store data. The registers which are used in the following syntactic and semantic description are defined:

Accumulator(AC) - 12-bit register in which arithmetic and logic operations are performed.

Multiplier quotient(MQ) - 12-bit biderectional shift register that acts as an extension of the AC.

Program counter(PC) - 12-bit register that is used to control the program sequence. The PC contains the address of the core location from which the next instruction is taken.

Link(L) - 1-bit register that is used to extend the arithmetic facilities of the AC.

Switch Register(SR) - 12-bit register which contains the status, bit by bit, of the data switches on the console.

Location Counter(LC) - 12-bit register which contains the address of the location of the next statement to be assembled.

In addition to the above registers, the following notation will be used:

Superscript - a↑v means a^v

Null set - NULL means $\emptyset$

Sets - [e] means set containg e

This notation was needed in order to utilize a constant character set. Other notations will be described as needed.

The syntax and semantics follow in three sections: Basics, consisting of constants, labels, etc; instructions; and program section.

This definition uses the following semantic attributes:

Value - the octal value of an integer quantity.

Length - the character length of an integer or alphanumeric.

Effect - the effect of an instruction, statement, etc., on the state of the computation. This state is a vector consisting of the registers and core memory locations.

Operation - a function which describes a computation whose result will be stored in a destination given by,

Destination Function - a function which gives the destination for the results of an operation.

Effective address - an inherited attribute which becomes a destination for the results of an operation.

Label Function - an inherited attribute giving the location of a symbolic address. This is actually a set of addresses and symbolic names in ordered pairs.

Exit Index - an attribute tells which instruction will be executed next.

State Location - tells where the statement is to be located.

CONSTANTS

This section defines:

Constants
Integers
Signed integers
Letters
Alphanumerics
Blanks
Comments
Address
Labels
Indirect bit

All digits and values are in octal. All semantic definition arithmetic in this and following sections is also octal so as to be compatible.

Note: I have not included a definition of special characters; ie. '.', ',', ':', etc., but an extension would be easy, and would also be included in the syntax for comments

```
<digit>a ::= '0'; b ::= '1'; c ::= '2'; d ::= '3'; e ::= '4';
     f ::= '5'; g ::= '6'; h ::= '7'

     <value>↑v
          a↑v = 0; b↑v = 1; c↑v = 2; d↑v = 3; e↑v = 4; f↑v = 5;
          g↑v = 6; h↑v = 7

<integer>x ::= <digit>a; y ::= <digit>b <integer>Z

     <length=↑n   (0 < n ≤ 4)
     <value>↑v
          x↑n = 1; x↑v = a↑v;
          y↑n = Z↑n + 1; y↑v = Z↑v + b↑v × exp(8, Z↑n)

<signed integer>x ::= <integer>a; y ::= '+' <integer>b;
     Z ::= '-' <integer>c

     <value>↑v
          x↑v = a↑v; y↑v = b↑v; Z↑v = -c↑v

<letter> ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' |
     'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' |
     't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'Z'

<alphanumeric>x ::= <letter>a; y ::= <alphanumeric>b <digit>c;
     Z ::= <alphanumeric>c <letter>e

     <length>↑n   (n ≤ 7)
          x↑n = 1; y↑n = b↑n + 1; Z↑n = c↑n + 1

<blank> ::= ' '

<comment> ::= '/' <letter> | '/' <digit> | <comment> <letter>
     <comment> <digit> | <comment> <blank>

<address> ::= <alphanumeric>

<label> ::= <alphanumeric> ','

<indirect bit>a ::= ' '; b ::= '1'

     <value>↑v
          a↑v = 0
          b↑v = 1
```

INSTRUCTIONS

In the PDP-8/E there are two major types of instructions: memory reference and operators. Memory reference instructions store or retrieve data from core. The operate instructions consist of three groups of microinstructions. Group 1 is principally for CLEAR, COMPLEMENT, ROTATE, and INCREMENT operations. Group 2 is used in checking contents of the Accumulator and Link, and continuing to, or skipping, the next instruction based on the check. Group 1 and Group 2 instructions are micro-progammable, that is instructions in each group may be combined with certain other instructions in the same group to form new instructions. These combinations are allowed only with instructions of a different sequence(order of execution) on the same group. Group 3 instructions are used to manipulate data between the MQ and AC registers.

In addition to these operate and memory reference instructions, there exist two others. The Load instruction is used for entering constants into AC. The '0' instruction simply saves space for a symbolic address.

In the memory reference instructions, indirect addressing is possible with a '1' in the inderction field. A blank results in direct addressing.

The PDP-8/E uses a paging address scheme, but users of PAL III assembly need not be concerned since proper page addressing is handled by the system.

<loader>x ::= 'nl' <integer>a '='

<effect>e
xte(s) = s' where s'(ac) = atv, s'(Z) = s(Z) for Z ≠ ac

<group 3 op>a ::= 'cam'; b ::= 'mqa'; c ::= 'mql'; d ::= 'swp'

<effect>te
ate(s) = s' where s'(ac) = 0; s'(mq) = 0; s'8Z) = s(Z) for Z ≠ ac and Z ≠ mq
bte(s) = s' where s'(ac) = s(ac) .or. s(mq), s'(Z) = s(Z) for Z ≠ ac
cte(s) = s' where s'(mq) = s(ac), s'(ac) = 0, s'(Z) = s(Z) for Z ≠ ac and Z ≠ mq
dte(s) = s' where s'(mq) = s(ac), s'(ac) = s(mq), s'(Z) = s(Z) for Z ≠ ac and Z ≠ mq

<gp2 seq 3>a ::= 'hlt'; b ::= 'osr'

<effect>te
ate(s) = s' where s'(run) = 0, s'(Z) = s(Z) for Z ≠ run
bte(s) = s' where s'(ac) = s(ac) .or. s(sr), s'(Z) = s(Z) for Z ≠ ac

<gp2 seq 2>a ::= 'cla'

<effect>te
ate(s) = s' where s'(ac) = 0, s'(Z) = s(Z) for Z ≠ ac

<gp2 seq 1>a ::= 'skp'; b ::= 'snl'; c ::= 'sZl'; d ::= 'sZa'; e ::= 'sna'; f ::= 'sma'; g ::= 'spa'

<effect>te
ate(s) = s' where s'(pc) = s(pc) + 1, s'(Z) = s(Z) for Z ≠ pc
bte(s) = s' where s'(pc) = if s(l) .eq. 1 then s(pc) + 1 else s(pc), s'(Z) = s(Z) for Z ≠ pc
cte(s) = s' where s'(pc) = if s(l) .eq. 0 then s(pc) + 1 else s(pc), s'(Z) = s(Z) for Z ≠ pc
dte(s) = s' where s'(pc) = if s(mac) .eq. 0 then s(pc) + 1 else s(pc), s'(Z) = s(Z) for Z ≠ pc
ete(s) = s' where s'(pc) = if s(ac) .ne. 0 then s(pc) + 1 else s(pc), s'(Z) = s(Z) for Z ≠ pc
fte(s) = s' where s'(pc) = if (s(ac) .and. 4000) .eq. 4000 then s(pc) + 1 else s(pc), s'(Z) = s(Z) for Z ≠ pc
gte(s) = s' where s'(pc) = if (s(ac) .and. 4000) .eq. 0 then s(pc) + 1 else s(pc), s'(Z) = s(Z) for Z ≠ pc

<gp1 seq 4>a ::= 'ral'; b ::= 'rtl'; c ::= 'rar'; d ::= 'rtr'; e ::= 'bsw'

<effect>te
ate(s) = s' where s'(ac) = s(ac)x2 + s(l), s'(l) = s(ac)/2↑13, s'(Z) = s(Z) for Z ≠ ac and Z ≠ l
bte(s) = ate(ate(s))
cte(s) = s' where s'(ac) = s(ac)/2↑13 + s(l)x2↑13, s'(l) = s(ac) .and. 4000, s'(Z) = s(Z) for Z ≠ ac and Z ≠ l

```
          dte(s) = cte(cte(s))
          ete(s) = s' where s'(ac) = s(ac)X2↑6 + (s(ac) .and. 7777)
               /2↑6, s'(Z) = s(Z) for Z ≠ ac

<gp1 seq 3>a ::= 'iac'; b ::= 'cia'

     <effect>te
          ate(s) = s' where s'(ac) = s(ac) + 1, s'(Z) = s(Z)
               Z ≠ ac
          bte(s) = s' where s'(ac) = ¬s(ac) + 1, s'(Z) = s(Z) for
               Z ≠ ac

<gp1 seq 2>a ::= 'cml'; b ::= 'cma'; c ::= 'cia'; d ::= 'stl';
     e ::= 'sta'

     <effect>te
          a-e(s) = s' where s'(l) = ¬s(l), s'(Z) = s(Z) for Z ≠ l
          bte(s) = s' where s'(ac) = ¬s(mcc), s'(Z) = s(Z) for
               Z ≠ ac
          cte(s) = bte(s) in <gp1 seq 3> definition above
          dte(s) = s' where s'(l) = 1, s'(Z) = s(Z) for Z ≠ l
          ete(s) = s' where s'(ac) = 7777, s'(Z) = s(Z) for Z ≠ ac

<gp1 seq 1>a ::= 'cll'; b ::= 'stl'; c ::= 'cla'; d ::= 'sta'

     <effect>te
          ate(s) = s' where s'(l) = 0, s'(Z) = s(Z) for Z ≠ l
          bte(s) = dte(s) in <gp1 seq 2> definition above
          cte(s) = s' where s'(ac) = 0, s'(Z) = s(Z) for Z ≠ ac
          dte(s) = ete(s) in <gp1 seq 2> definition above

<group 2 op>a ::= <gp2 seq 1>m; b ::= <gp2 seq 2>n;
     c ::= <gp2 seq 3>o; d ::= <gp2 seq 1>p <gp2 seq 2>q;
     e ::= <gp2 seq 1>r <gp2 seq 3>s;
     f ::= <gp2 seq 2>t <gp2 seq 3>u

     <effect>te
          ate(s) = wte(s)
          bte(s) = nte(s)
          cte(s) = ote(s)
          dte(s) = qte(pte(s))
          ete(s) = ste(rte(s))
          fte(s) = ute(tte(s))
          gte(s) = xte(wte(vte(s)))

<group 1 op>a ::= 'nop'; b ::= <gp1 seq 1>b1; c ::= <gp1 seq 2>c1;
     d ::= <gp1 seq 3>d1; e ::= <gp1 seq 4>e1;
     f ::= <gp1 seq 1>f1 <gp1 seq 2>f2;
     g ::= <gp1 seq 1>g1 <gp1 seq 3>g2;
     h ::= <gp1 seq 1>h1 <gp1 seq 4>h2;
     i ::= <gp1 seq 2>i1 <gp1 seq 3>i2;
     j ::= <gp1 seq 2>j1 <gp1 seq 4>j2;
     k ::= <gp1 seq 3>k1 <gp1 seq 4>k2;
     l ::= <gp1 seq 1>l1 <gp1 seq 2>l2 <gp1 seq 3>l3;
     m ::= <gp1 seq 1>m1 <gp1 seq 2>m2 <gp1 seq 4>m3;
     n ::= <gp1 seq 1>o1 <gp1 seq 3>n2 <gp1 seq 4>n3;
     o ::= <gp1 seq 2>o1 <gp1 seq 3>o2 <gp1 seq 4>o3;
     p ::= <gp1 seq 1>p1 <gp1 seq 2>p2 <gp1 seq 3>p3 <gp1 seq 4>p4
```

```
     <effect>↑e
          a↑e(s) = s
          b↑e(s) = b1↑e(s)
          c↑e(s) = c1↑e(s)
          d↑e(s) = d1↑e(s)
          e↑e(s) = e1↑e(s)
          f↑e(s) = e2↑e(e1↑e(s))
          g↑e(s) = g2↑e(g1↑e(s))
          h↑e(s) = h2↑e(h1↑e(s))
          i↑e(s) = i2↑e(i1↑e(s))
          j↑e(s) = j2↑e(j1↑e(s))
          k↑e(s) = k2↑e(k1↑e(s))
          l↑e(s) = l3↑e(l2↑e(l1↑e(s)))
          m↑e(s) = m3↑e(m2↑e(m1↑e(s)))
          n↑e(s) = n3↑e(n2↑e(n1↑e(s)))
          o↑e(s) = o3↑e(o2↑e(o1↑e(s)))
          p↑e(s) = p4↑e(p3↑e(p2↑e(p1↑e(s))))

<memory opcode type 1>a ::= 'and'; b ::= 'tad'

     <opration>↑o
     <effective address>↑a
     <destination fcn>↑r
          a↑o(s,a↑a) = s(ac) .and. s(a↑a), a↑r = true
          b↑o(s,b↑a) = s(ac) + s(b↑a), b↑r = true

<memory opcode type 2>a ::= 'isZ'

     <operation fcn>↑o
     <effective address>↑a
     <destination fcn>↑r
          a↑o(s,a↑a) = s(a↑a) + 1, a↑r = false

<memory opcode type 3>a ::= 'dca'

     <operation fcn>↑o
     <effective address>↑a
     <destination fcn>↑r
          a↑o(s,a↑a) = s(ac), a↑r = false

<memory ref instructiona ::= <memory opcode type 1>u
     <indirect bit>x <address>l;
     b ::= <memory opcode type 2>v <indirect bit>y <address>m;
     c ::= <memory opcode type 3>w <indirect bit>Z <address>n;
     d ::= 'jms' <address>o; e ::= 'jmp' <address>p

     <label fcn>↑l
     <effective address>↑a
     <destination fcn>↑r
     <destination>↑d
     <operation fcn>↑o
     <effect>↑e
     <value>↑v
          a↑e(s) = s' where s'(a↑a) = u↑o(s,a↑a), s'(Z) = s(Z) for
               Z ≠ ac
          a↑a(s) = if x↑v .eq. 0 then l↑l(s) else s(l↑l)
          a↑d = if u↑r then ac else a↑a(s)
          b↑e(s) = s' where s'(b↑a) = v↑o(s,b↑a), s'(pc) = if
               v↑o(s,b↑a) .eq. 0 then s(pc) + 1 else s(pc), s'(Z) =
```

```
            s(Z) for y ≠ m↑1 and Z ≠ pc
        b↑a(s) = if y↑v .eq. 0 then m↑1(s) else s(m↑1)
        b↑d = if v↑r then ac else b↑a(s)
        c↑e(s) = s' where s'(c↑a) = w↑o(s,c↑a), s'(ac) = 0, s'(Z)
            = s(Z) for Z ≠ n↑1 and Z ≠ ac
        c↑a(s) = if Z↑v .eq. 0 then n↑1(s) else s(n↑1)
        c↑d = if v↑r then ac else c↑a(s)
        d↑e(s) = s' where s'(pc) = s(o↑1) + 1, s'(o↑1) = pc,
            s'(Z) = s(Z) for Z ≠ pc and Z ≠ o↑1
        e↑e(s) = s' where s'(pc) = s(p↑1), s'(Z) = s(Z) for Z
            ≠ pc

<instruction>a ::= <memory ref instruction>m; b ::= <group 1 op>n;
    c ::= <group 2 op>o; d ::= <group 3 op>p; e ::= <loader>q;
    f ::= '0'

    <effect>↑e
        a↑e(s) = m↑e(s)
        b↑e(s) = n↑e(s)
        c↑e(s) = o↑e(s)
        d↑e(s) = p↑e(s)
        e↑e(s) = q↑e(s)
        f↑e(s) = null
```

PROGRAM

This section defines a program to consist of:

1. Location Codes - Specifies where the program is stored
2. Body - The Statements consisting of Instructions and Comments which are the actual program
3. End - the final statement which tells the assembler it is done.

<statement>a ::= <instruction>m; b ::= <comment>;
c ::= <label>i <instruction>n; d ::= <statement>e <comment>

<label fcn>↑l
<exit index>↑x
<effect>↑e
<state location>↑m
a↑l = null
a↑x = s(pc) + 1
a↑e(s) = m↑e(s)
a↑m = s(lc)
b↑l = null
b↑e(s) = null
b↑m = s(lc)
c↑l = [i]
c↑x = s(pc) + 1
c↑e(s) = n↑e(s)
c↑m = s(lc)
d↑l = e↑l
d↑x = e↑x
d↑l(s) = e↑e(s)
d↑m = e↑m

<body>a ::= <statement>x; b ::= <body>c <statement>y

<label fcn>↑l
<effect>↑e
<state location>↑m
a↑l = x↑l
a↑e(s) = x↑e(s)
a↑m = x↑m
b↑l = c↑l .union. y↑l
b↑e(s) = y↑e(c↑e(s))

<end>a ::= '$'

<effect>↑e
<state location>↑m
a↑e(s) = .stop assembly.
a↑m(s) = s(lc)

<location code>a ::= '*' <integer>x

<effect>↑e
<value>↑v
a↑e(s) = s' where s'(lc) = x↑v

<program>a ::= <location code>b <body>c <end>d

<effect>↑e
<lable fcn>↑l
a↑e(s) = d↑e(c↑e(b↑e(s)))
a↑l(s) = c↑l(s)

REFERENCES

Digital Equipment Corporation, PDP-8/E Small Computer Handbook, 1970.

Elgot, C. C., and Robinson, Random-Access Stored Program Machines, an Approach to Programming Languages, Journal of the ACM 11, 4, 1964.

Maurer, W. D., Introduction to Programming Science Part I, Electronics Research Laboratory, University of California, Berkeley, 1972.

Wagner, E. G., Bounded Action Machines: Toward an Abstract Theory of Computer Structure, Computer and System Science, 2, 1, 1968.

C H A P T E R F I V E

A D E S C R I P T I O N O F B P L

(Burroughs' systems programming language)

B Y

W A L T L A P I N S K Y

B P L

BURROUGHS PROGRAMMING LANGUAGE

FOR THE BURROUGHS MEDIUM SYSTEMS COMPUTERS

B2500, B3500, AND B4700

INTRODUCTION

THIS SECTION OF THE PAPER IS INTENDED AS A BRIEF INTRODUCTION TO THE MACHINE AS SEEN BY THE USER WHEN WRITTING IN BPL. IT IS NOT INTENDED TO BE COMPLETE; THE DEFINITION AND SEMANTICS OF THE LANGUAGE WHICH FOLLOW SHOULD EXACTLY DESCRIBE THE VIRTUAL MACHINE. THE PURPOSE OF THIS SECTION IS TO GIVE THE READER AN UNDERSTANDING OF THE BASIC HARDWARE WHICH THE LANGUAGE IS IMPLEMENTED ON.

THE VIRTUAL MACHINE THAT A USER (POSSIBLY ONE OF SEVERAL SIMULTANEOUS USERS) SEES IN BPL CONSISTS OF A CONSECUTIVE STRING OF INDIVIDUALLY ADDRESSABLE DIGITS, STARTING WITH 0. EACH DIGIT IS 4 BITS LONG. THE VALUE OF A PARTICULAR BIT CAN BE CHECKED AND CHANGED IF DESIRED. THE BITS ARE IDENTIFIED BY THEIR CORRESPONDING BINARY VALUE: 1 BIT, 2 BIT, 4 BIT, AND 8 BIT. IN ADDITION, PAIRS OF DIGITS STARTING WITH AN EVEN ADDRESS DIGIT MAY BE CONSIDERED AS A CHARACTER OR BYTE. A BYTE IS THEREFORE 8 BITS LONG AND CAN BE ANY OF THE POSSIBLE 256 EBCDIC CHARACTERS. IN ADDITION, FOUR DIGITS STARTING WITH A MOD 4 ADDRESS ARE CONSIDERED A WORD. THE HARDWARE CONTAINS INSTRUCTIONS FOR MANULIPULATING DIGITS, CHARACTERS, AND WORDS.

ANY SINGLE ELEMENT OF A USER-S DATA STRUCTURE HAS THREE ATTRIBUTES: ADDRESS, SIZE, AND TYPE. EXCEPT FOR CERTAIN SPECIAL CASES, THE ADDRESS OF A DATA ELEMENT IS DETERMINED BY THE COMPILER IN THE USUAL WAY (IN THE ORDER SPECIFIED BY THE USER IN HIS SOURCE). THE ADDRESS IS ALWAYS THE ADDRESS OF THE LOWEST ADDRESSED DIGIT IN A PARTICULAR ELEMENT. FOR EXAMPLE, IF THE ELEMENT CTR OCCUPIES 6 DIGITS ADDRESSED FROM 32 THROUGH 37 INCLUSIVE, THE ADDRESS OF CTR WOULD BE 32. THE SIZE OF AN ELEMENT IS THE NUMBER OF DIGITS OR CHARACTERS IN THE FIELD. THE CHOICE OF DIGITS OR CHARACTERS DEPENDS ON THE TYPE. A DATA ELEMENT CAN HAVE ANY OF THE FOLLOWING TYPES:

INTEGER — COMPOSED OF A CONSECUTIVE STRING OF DIGITS EACH CONTAINING ONLY VALID NUMBERS (0 - 9). SIZE EXPRESSED IN DIGITS.

REAL — A REAL NUMBER IN THE FORTRAN F FORMAT SENSE. SIZE EXPRESSED IN DIGITS.

SIGNED — SAME AS INTEGER, BUT ALSO CONTAINS A SIGN. SIZE EXPRESSED IN DIGITS.

COMP STRING — COMPOSED OF A CONSECUTIVE STRING OF DIGITS, EACH OF WHICH CAN BE ANY OF THE POSSIBLE 16 BIT COMBINATIONS. SIZE EXPRESSED IN DIGITS. NON-INTEGER VALUES ARE CALLED "UNDIGITS" AND ARE REFERED TO AS A THROUGH F (A = 1010(2), B = 1011(2), ..., F = 1111(2)).

NUMERIC — COMPOSED OF A CONSECUTIVE STRING OF CHARACTERS, EACH OF WHICH IS A VALID DIGIT ONLY ("0", "1", ..., "9"). SIZE EXPRESSED IN CHARACTERS.

ALPHA — COMPOSED OF A CONSECUTIVE STRING OF CHARACTERS, EACH OF WHICH CAN BE ANY OF THE POSSIBLE 256 EBCDIC CHARACTERS. SIZE EXPRESSED IN CHARACTERS.

BOOLEAN — A SINGLE BIT USED AS A BOOLEAN TRUE/FALSE SWITCH. (BIT IS SET IF TRUE, RESET IF FALSE.) SIZE IS 1 BIT. UP TO FOUR BOOLEAN VARIABLES CAN BE PLACED IN A SINGLE DIGIT BY THE COMPILER.

THE EXACT DEFINITION AND USAGE OF THE VARIOUS TYPES WILL BE DESCRIBED IN THE SYNATX AND SEMANTICS. THE FOLLOWING ADDITIONAL NOTES ARE DIVIDED INTO TWO PARTS. PART 1 DEALS WITH ADDITIONAL COMMENTS ABOUT THE VIRTUAL MACHINE. PART 2 DEALS WITH CONVENTIONS THAT WILL BE FOLLOWED IN THIS PAPER.

ADDITIONAL NOTES, PART 1:

1. ARITHMETIC IS ONLY DEFINED FOR TYPES INTEGER, NUMERIC, OR SIGNED (MIXED IN ANY MANNER WITH HARDWARE CONVERSION AUTOMATICALLY TAKING PLACE) OR BETWEEN REAL FIELDS.

2. INDIRECT ADDRESSING IS PERMITTED. AN ADDRESS IS 6 DIGITS LONG AND CONSISTS OF A CONTROLLER (WHICH INDICATES INDEXING AND TYPE) AND THE ADDRESS.

3. THE VIRTUAL MACHINE CONTAINS SEVERAL REGISTER-LIKE ITEMS WHICH THE USER MAY USE, BUT WHICH ACTUALLY RESIDE IN HIS ADDRESS SPACE. THESE INCLUDE THREE INDEX REGISTERS, "IX1", "IX2", AND "IX3". THEY ALLOW ARRAY TYPE ACCESSING WHEN ARRAYS ARE NOT SPECIFIED. ADDITIONAL AREAS OF USER'S MEMORY ARE AFFECTED BY SUBROUTINE CALLS (STACK POINTERS) AND VARIOUS TESTING INSTRUCTIONS. THESE WILL ALL BE DESCRIBED IN THEIR PLACE. INDEXING AND INDIRECTION CAN BE CARRIED OUT IN ANY DESIRED ORDER AND TO ANY DESIRED LEVEL.

4. THERE IS A COMPARISON INDICATOR WHOSE VALUE IS CHANGED BY VARIOUS INSTRUCTIONS. IT CAN BE TESTED BUT DOES NOT EXIST IN THE USER'S MEMORY. IT WILL BE SHOWN AS CMP[] IN THIS PAPER. IT CAN TAKE ON ANY OF "EQUAL", "LESS", OR "GREATER".

ADDITIONAL NOTES, PART 2

1. A LIST OF RESERVED WORDS IS PROVIDED AS APPENDIX A. NO IDENTIFIER DEFINED BY THE USER CAN BE ANY OF THESE RESERVED WORDS EXCEPT AS NOTED.

2. A LIST OF EQUIVALENT FORMS IS PROVIDED IN APPENDIX B. THE SYNTAX AND SEMANTIC DEFINITIONS THAT FOLLOW WILL ONLY USE ONE FORM OF A SYNTAX ELEMENT. ANY OF THE EQUIVALENT FORMS CAN BE USED AT ANY POINT WHERE THE SPECIFIED LANGUAGE ELEMENT IS USED.

3. APPENDIX C DEALS WITH DEFINES. A DEFINE IS A METHOD OF DEALING WITH VARIOUS FORMS OF SHORTHAND TO THE COMPILER AND DO NOT AFFECT THE ACTUAL SYNTAX OF THE LANGUAGE.

4. THE TERMINAL <EMPTY> IS DEFINED TO BE THE STRING CONTAINING NO CHARACTERS OR SYMBOLS.

5. WE SHALL USE THE BINARY OPERATOR : TO INDICATE THE CONCATENATION OF TWO DIGIT OR CHARACTER STRINGS TO FORM ONE DIGIT OR CHARACTER STRING. THIS IS TO DISTINGUISH BETWEEN THE DEFINITION OF THINGS LIKE INTEGERS WHERE THE VALUE OF AN INTEGER IS IN ONE CASE THE VALUE OF AN ARITHMETIC EXPRESSION ON THE VALUE OF ANOTHER INTEGER AND A DIGIT, AND A CHARACTER STRING WHICH IS FORMED AS A COMBINATION OF CHARACTERS (AND HAS NO ARITHMETIC VALUE).

6. DEFINE CONTENT[ADR, TYP, SIZ] TO BE THE CONTENTS OF THE USER-S MEMORY DEFINED AS STARTING AT DIGIT ADR, OF TYPE TYP FOR A SIZE OF SIZ.

7. <INDETERMINABLE> IS THE VALUE OF AN ITEM WHOSE ACTUAL VALUE IS UNKNOWN.

8. DEFINE COMP[X] TO BE "EQUAL" IF X = 0, "LESS" IF X < 0, OR "GREATER" IF X > 0. IT IS ONLY DEFINED FOR X AN ARITHMETIC EXPRESSION.

9. DUE TO THE PRINTER LIMITATIONS, THE EXACT FORM OF PROFESSOR MAURER-S PAPER CAN NOT BE USED (SEE REFERENCE 2). THE FOLLOWING SYMBOL CONVENTIONS WILL BE FOLLOWED INSTEAD:

   A. INSTEAD OF SUPERSCRIPTS, FUNCTIONS WITH SQUARE BRACKETS WILL BE USED. FOR EXAMPLE, VALUE[A] INSTEAD OF A WITH A SUPERSCRIPT V. THIS WILL BE CARRIED OUT TO ANY DESIRED NESTING LEVEL.

   B. THE DOLLAR SIGN WILL BE USED IN PLACE OF THE SINGLE QUOTE TO INDICATE ACTUAL CHARACTER STRINGS. THUS A IS THE STRING CONTAINING THE CHARACTER "A". (NOTE: EXCEPT AS COMPILER COMMANDS, "$" DOES NOT APPEAR IN THE BPL LANGUAGE.)

   C. <: WILL BE USED TO INDICATE SET THEORY "IS CONTAINED IN".

   D. :(AND): WILL BE USED FOR THE CURLY BRACKETS DEFINING THE ELEMENTS OF A SET.

   E. .U. WILL BE USED FOR THE SET UNION OPERATOR.

   F. // WILL BE USED FOR THE VERTICAL LINE INDICATING ALTERNATION.

[illegible]

[illegible]

[illegible]

INPUT FORMAT

THE SOURCE INPUT TO BPL CONSISTS OF CARD-IMAGE RECORDS. THE FIRST 72 COLUMNS OF EACH CARD-IMAGE RECORD ARE AVAILABLE FOR SOURCE DATA (WITH NO SPECIAL COLUMN MEANINGS); COLUMNS 73-80 INCLUSIVE ARE AVAILABLE FOR OPTIONAL SEQUENCE INFORMATION. COLUMN 72 OF A RECORD IS FOLLOWED IMMEDIATELY BY COLUMN 1 OF THE NEXT RECORD: END OF SOURCE-RECORD HAS NO SYNTAX SIGNIFICANCE (EXCEPT IN A COMMENT).

COMMENTS ARE PLACED BY USE OF THE SYMBOL "&" FOLLOWED BY ANY CHARACTER STRING AND CONTINUES TO THE END OF THAT RECORD (WHICH CONTAINS THE "&"). IN THIS CASE, THE COMMENT DOES SERVE AS A SEPERATOR. WE SHOW THIS AS FOLLOWS:

<COMMENT> ::= && <ANY STRING OF CHARACTERS UP THRU COLUMN 72>

<FILLER> ::= <EMPTY> // <FILLER> $ $ // <FILLER> <COMMENT>

<BLANKS> ::= <FILLER> $ $ // <FILLER> <COMMENT>

IN ORDER TO PRESERVE AS MUCH READIBILITY AS POSSIBLE IN THE DEFINITIONS TO FOLLOW, WHENVER A SINGLE BLANK APPEARS IN A STRING WITHIN $-S, THAT BLANK IS TO BE ASSUMED TO BE <BLANKS>. FOR EXAMPLE,

$IF $ IS TO BE TAKEN AS IF <BLANKS>.

ADDITIONAL PLACES FOR ARBITRARY SPACING OR COMMENTS ARE DEFINED IN APPENDIX B, EQUIVALENCES. FOR EXAMPLE,

$+$ CAN BE TAKEN AS <FILLER> $+$ <FILLER>.

CONSTANTS

```
<DIGIT>A ::= $0$;  B ::= $1$;  C ::= $2$;  D ::= $3$;  E ::= $4$;
     F ::= $5$;  G ::= $6$;  H ::= $7$;  I ::= $8$;  J ::= $9$

     VALUE[A] = 0;  VALUE[B] = 1;  VALUE[C] = 2;  VALUE[D] = 3;
          VALUE[E] = 4;  VALUE[F] = 5;  VALUE[G] = 6;
          VALUE[H] = 7;  VALUE[I] = 8;  VALUE[J] = 9


<INTEGER>I ::= <DIGIT>D;
     J ::= <INTEGER>K <DIGIT>L

     VALUE[I] = VALUE[D];  VALUE[J] = 10×(VALUE[K]) + VALUE[L]

     LENGTH[I] = 1;  LENGTH[J] = LENGTH[K] + 1
          (LENGTH[K] < 99)

     TYPE[I] = INTEGER


<SIGNED INTEGER>S ::= $+$ <INTEGER>I;
     T ::= $-$ <INTEGER>J

     VALUE[S] = VALUE[I];  VALUE[T] = -VALUE[J]

     LENGTH[S] = LENGTH[I];  LENGTH[T] = LENGTH[J]

     TYPE[S] = SIGNED;  TYPE[T] = SIGNED


<REAL NUMBER>R ::= <INTEGER>I $.$ <INTEGER>J

     VALUE[R] = VALUE[I] + (VALUE[J]/EXP(10, LENGTH[J]))

     LENGTH[K] = LENGTH[I] + LENGTH[J]
          (LENGTH[R] ≤ 99)

     TYPE[R] = REAL


<SIGNED REAL>S ::= $+$<REAL NUMBER>R;
     T ::= $-$<REAL NUMBER>U

     VALUE[S] = VALUE[R];  VALUE[T] = -VALUE[U]
```

```
      LENGTH[S] = LENGTH[R]; LENGTH[T] = LENGTH[U]

      TYPE[S] = REAL; TYPE[T] = REAL

<UNDIGIT>A ::= $A$; B ::= $B$; C ::= $C$; D ::= $D$;
      E ::= $E$; F ::= $F$

      VALUE[A] = 10; VALUE[B] = 11; VALUE[C] = 12;
            VALUE[D] = 13; VALUE[E] = 14; VALUE[F] = 15

<HEX DIGIT>H ::= <UNDIGIT>U; I ::= <DIGIT>D

      VALUE[H] = VALUE[U]; VALUE[I] = VALUE[D]

<HEX NUMBER>H ::= <HEX DIGIT>D;
      I ::= <HEX NUMBER>J <HEX DIGIT>E

      VALUE[H] = VALUE[D]; VALUE[I] = VALUE[J] : VALUE[E]

      LENGTH[H] = 1; LENGTH[I] = LENGTH[J] + 1

<COMPUTATIONAL HEX NUMBER>C ::= $@$ <HEX NUMBER>H $@$

      VALUE[C] = VALUE[H]

      LENGTH[C] = LENGTH[H]
            (LENGTH[H] ≤ 99)

      TYPE[C] = COMP STRING

<DISPLAY HEX NUMBER>D ::= $%$ <HEX NUMBER>H $%$

      VALUE[D] = VALUE[H]

      LENGTH[D] = LENGTH[H]/2
            (LENGTH[H] = 0 MOD 2, LENGTH[H] ≤ 198)

      TYPE[D] = ALPHA

<STRING CHARACTER>S ::= <ANY SINGLE ONE OF THE 256 EBCDIC
```

CHARACTERS (INCLUDING BLANK), EXCEPT @9F@ AND $"$>A;
T ::= $""$

VALUE[S] = VALUE[A]; VALUE[T] = $"$

LENGTH[S] = 1; LENGTH[T] = 1

<STRING>S ::= <STRING CHARACTER>C;
T ::= <STRING>U <STRING CHARACTER>D

VALUE[S] = VALUE[C]; VALUE[T] = VALUE[U] : VALUE[D]

LENGTH[S] = 1; LENGTH[T] = LENGTH[U] + LENGTH[D]

<ALPHA CONSTANT>A ::= $"$ <STRING>S $"$

VALUE[A] = VALUE[S]

LENGTH[A] = LENGTH[S]
(LENGTH[S] ≤ 99)

TYPE[A] = ALPHA

<LABEL LITERAL>L ::= $[$ <VARIABLE>V $]$

VALUE[L] = ADDRESS[V]

LENGTH[L] = 6

TYPE[L] = INTEGER

(IT SHOULD BE NOTED THAT THE ADDRESS[V] CONTAINS INFORMATION ABOUT THE TYPE[V] AND ANY INDEXING AND/OR INDIRECTION DESIRED. THE EXACT FORM OF THIS INFORMATION IS DESCRIBED UNDER DECLARATIONS BELOW).

<BOOLEAN CONSTANT>B ::= $TRUE$; C ::= $FALSE$

VALUE[B] = TRUE; VALUE[C] = FALSE

LENGTH[B] = 1 BIT; LENGTH[C] = 1 BIT

```
      TYPE[B] = BOOLEAN;  TYPE[C] = BOOLEAN

<ARITHMETIC CONSTANT>A ::= <INTEGER>I;  B ::= <REAL NUMBER>R;
      C ::= <LABEL LITERAL>L

      VALUE[A] = VALUE[I];  VALUE[B] = VALUE[R];  VALUE[C] =
           VALUE[L]

      LENGTH[A] = LENGTH[I];  LENGTH[B] = LENGTH[R];  LENGTH[C] =
           LENGTH[L]

      TYPE[A] = INTEGER;  TYPE[B] = REAL;  TYPE[C] = INTEGER


<SIGNED ARITHMETIC CONSTANT>S ::= <ARITHMETIC CONSTANT>A;
      T ::= <SIGNED INTEGER>I;  U ::= <SIGNED REAL>R

      VALUE[S] = VALUE[A];  VALUE[T] = VALUE[I];  VALUE[U] =
           VALUE[R]

      LENGTH[S] = LENGTH[A];  LENGTH[T] = LENGTH[I];  LENGTH[U] =
           LENGTH[R]

      TYPE[S] = TYPE[A];  TYPE[T] = SIGNED;  TYPE[U] = REAL


<SIMPLE CONSTANT>A ::= <ARITHMETIC CONSTANT>T;
      B ::= <COMPUTATIONAL HEX NUMBER>U;
      C ::= <DISPLAY HEX NUMBER>V;  D ::= <ALPHA CONSTANT>W

      VALUE[A] = VALUE[T];  VALUE[B] = VALUE[U];  VALUE[C] =
           VALUE[V];  VALUE[D] = VALUE[W]

      LENGTH[A] = VALUE[T];  LENGTH[B] = VALUE[U];  LENGTH[C] =
           VALUE[V];  LENGTH[D] = VALUE[W]

      TYPE[A] = TYPE[T];  TYPE[B] = COMP STRING;  TYPE[C] = ALPHA;
           TYPE[D] = ALPHA


<SIGNED CONSTANT>A ::= <SIMPLE CONSTANT>T;
      B ::= <SIGNED ARITHMETIC CONSTANT>U

      VALUE[A] = VALUE[T];  VALUE[B] = VALUE[U]
```

```
LENGTH[A] = LENGTH[T];  LENGTH[B] = LENGTH[U]

TYPE[A] = TYPE[T];  TYPE[B] = TYPE[U]
```

THERE ARE CERTAIN TYPES OF CONSTANTS WHOSE ACTUAL ATTRIBUTES DEPEND ON THEIR USE. IN EACH CASE, THE ATTRIBUTE DEPENDS ON THE FIELD RECEIVING THE CONSTANT. WE INTRODUCE THE INHERITED ATTRIBUTE
RSIZE[] TO BE THE SIZE OF THE RECEIVING FIELD.

```
<SPECIAL CONSTANT>A ::= $[$ $ALL$ $]$ <SIMPLE CONSTANT>S;
     B ::= $[$ $JSR$ $]$ <SIMPLE CONSTANT>T;
     C ::= $[$ $JSL$ $]$ <SIMPLE CONSTANT>U
```

LET REPEAT[X, Y] BE THE FIELD FORMED BY REPEATING X A SUFFICIENT NUMBER OF TIMES SUCH THAT THE LENGTH OF THE CONSTRUCTED FIELD IS Y UNITS LONG (UNITS DIGITS OR CHARACTERS DEPENDING ON TYPE[x]). FOR EXAMPLE, REPEAT["A", 5] = "AAAAA".

```
VALUE[A] = REPEAT[VALUE[S], RSIZE[] ]
     (RSIZE[] > LENGTH[S])
```

LET FILL[N, T] BE THE FIELD FORMED BY N CONSECUTIVE SPACES IF T = ALPHA; OTHERWISE N CONSECUTIVE DIGIT ZEROS.

```
VALUE[B] = FILL[RSIZE[] - LENGTH[T], TYPE[T] ]  :  VALUE[T];
VALUE[C] = VALUE[U]  :  FILL[RSIZE[] - LENGTH[U], TYPE[U] ]
     (LENGTH[T] < RSIZE[];  LENGTH[U] < RSIZE[])

LENGTH[A] = RSIZE[];  LENGTH[B] = RSIZE[];  LENGTH[C] =
     RSIZE[]

TYPE[A] = TYPE[S];  TYPE[B] = TYPE[T];  TYPE[C] = TYPE[U]
```

EXAMPLES: IF A1, A2, A3 ARE EACH DECLARED TO BE OF TYPE ALPHA AND OF LENGTH 8, THEN

```
     A1 ← [ALL] "AB";  A2 ← [JSR] "AB";  A3 ← [JSL] "AB"
```

WOULD RESULT IN

```
     A1 = "ABABABAB",  A2 = "      AB",  A3 = "AB      "
```

```
<CONSTANT>A ::= <SIGNED CONSTANT>S;  B ::= <SPECIAL CONSTANT>T;
     C ::= <BOOLEAN CONSTANT>U

     VALUE[A] = VALUE[S];  VALUE[B] = VALUE[T];  VALUE[C] =
          VALUE[U]

     LENGTH[A] = LENGTH[S];  LENGTH[B] = LENGTH[T];  LENGTH[C] =
          LENGTH[U]

     TYPE[A] = TYPE[S];  TYPE[B] = TYPE[T];  TYPE[C] = TYPE[U]
```

IDENTIFIERS

<LETTER> ::= A // B // C // D // E // F // G //
H // I // J // K // L // M // N // O //
P // Q // R // S // T // U // V // W //
X // Y // Z

<NAME>N ::= <LETTER>L; M ::= <NAME>P <LETTER>Q

LENGTH[N] = 1; LENGTH[M] = LENGTH[P] + 1
(LENGTH[P] ≤ 29)

ADDRESS[]

TYPE[]

SIZE[]

VALUE[]

(<NAME> MAY NOT BE A RESERVED WORD EXCEPT $IX1$, $IX2$,
$IX3$, OR $BASE$. SEE APPENDIX A. THE ATTRIBUTES OF
THESE SPECIAL NAMES ARE:
ADDRESS[$IX1$] = 8; SIZE[$IX1$] = 7; TYPE[$IX1$] =
SIGNED
ADDRESS[$IX2$] = 16; SIZE[$IX2$] = 7; TYPE[$IX2$] =
SIGNED
ADDRESS[$IX3$] = 24; SIZE[$IX3$] = 7; TYPE[$IX3$] =
SIGNED
ADDRESS[$BASE$] = 0; SIZE[$BASE$] = 1; TYPE[$BASE$] =
INTEGER)

FROM THIS POINT ON (S) WILL BE USED TO SHOW A FUNCTION
OF THE STATE VECTOR S. SIDE[] IS THE SIDE EFFECT OF A
SYNTAX ELEMENT.

<VARIABLE>V ::= <NAME>N; W ::= <NAME>X <MODIFIER>M

```
ADDRESS[V] = ADDRESS[N];
ADDRESS[W] = ADDRESS[X] + ADDRESS[M]

TYPE[V] = TYPE[N];  TYPE[W] = IF TYPE[M] = NONE THEN TYPE[X]
     ELSE TYPE[M]
     (TYPE[V] ≠ BOOLEAN;  TYPE[W] ≠ BOOLEAN)

SIZE[V] = SIZE[N];  SIZE[W(S)] = IF SIZE[M] = NONE THEN
     SIZE[X] ELSE SIZE[M(S)]

VALUE[V] = VALUE[N];  VALUE[W(S)] = IF TYPE[W] = INDIRECT
     THEN
     CONTENT[CONTENT[ADDRESS[W], INTEGER, 6], TYPE[W],
     SIZE[W(S)]]  ELSE
     CONTENT[ADDRESS[W], TYPE[W], SIZE[W(S)]]

NTYPE[]     (THE TYPE OF THE VARIABLE NAME WITHOUT ANY
     MODIFIERS [IE:  THE DECLARED TYPE])

NSIZE[]     (THE SIZE OF THE VARIABLE NAME WITHOUT ANY
     MODIFIERS [IE:  THE DECLARED SIZE])


<MODIFIER>M ::= <TYPE MODIFIER>T <INCREMENT MODIFIER>I <INDEX
     MODIFIER>X <SIZE MODIFIER>Z

          (NOTE:  A <MODIFIER> ALWAYS EXISTS FOR A PARTICULAR
          <NAME>.  WE USE THE INHERITED ATTRIBUTE NTYPE[] AS
          DEFINED ABOVE.  THE ORDER OF THE INDIVIDUAL MODIFIERS
          WITHIN <MODIFIER> IS UNIMPORTANT AND DOES NOT AFFECT
          THE RESULT.)

     NTYPE[]

     ADDRESS[M] = ADDRESS[I] + ADDRESS[X]

     TYPE[M] = TYPE[T]

     SIZE[M(S)] = SIZE[Z(S)]

     WITHIN A <MODIFIER> (ONLY), ANY TIME $.$ APPEARS, IT CAN BE
     REPLACED BY
          <FILLER> $.$ <FILLER>.


<TYPE MODIFIER>A ::= <EMPTY>;  B ::= $.$ $UNS;  C ::= $.$ $UAS;
```

```
D ::= $.$ $SN$;  E ::= $.$ $IA$;  F ::= $.$ $NM$;
G ::= $.$ $NO$

TYPE[A] = NONE;  TYPE[B] = INTEGER;  TYPE[C] = ALPHA;
     TYPE[D] = SIGNED;  TYPE[E] = INDIRECT;
     TYPE[F] = NUMERIC;  TYPE[G] = INTEGER


<INCREMENT MODIFIER>I ::= <EMPTY>;  J ::= $.$ <SIGNED INTEGER>K

     (LENGTH[K] ≤ 6)

ADDRESS[I] = 0;
     ADDRESS[J] = IF (NTYPE[] = ALPHA OR NTYPE[] = NUMERIC)
     THEN 2×VALUE[K] ELSE VALUE[K]


<INDEX MODIFIER>X ::= <EMPTY>;  A ::= $.$ $IX1$;
B ::= $.$ $IX2$;  C ::= $.$ $IX3$

ADDRESS[X] = 0;  ADDRESS[A] = VALUE[$IX1$];  ADDRESS[B] =
     VALUE[$IX2$];  ADDRESS[C] = VALUE[$IX3$]


<SIZE MODIFIER>Z ::= <EMPTY>;  T ::= $.$<INTEGER>I;
U ::= $.$<NAME>N

SIZE[Z(S)] = NONE;  SIZE[T(S)] = VALUE[I]
     (LENGTH[I] ≤ 6)
SIZE[U(S)] = IF (SIZE[N] = 2 AND ADDRESS[N] = 0 MOD 2 AND
     ADDRESS[N] ≤ 38) THEN VALUE[N(S)] ELSE SIZE[N]
     (N ≠ $IX1$;  N ≠ $IX2$;  N ≠ $IX3$)

     (THIS CONSTRUCT WHERE ADDRESS[N] IS USED AS THE SIZE OF
     A FIELD IS CALLED "INDIRECT FIELD LENGTH").


<SUBSCRIPT>T ::= <INTEGER>I;  U ::= <VARIABLE>V

VALUE[T] = VALUE[I];  VALUE[T] = VALUE[V]
     (TYPE[V] = INTEGER OR SIGNED OR NUMERIC)
```

ARRAYS: BPL ONLY ALLOWS ONE DIMENSIONAL ARRAYS. BY THE SIZE[A] OF AN ARRAY A WE SHALL MEAN THE SIZE OF ONE ELEMENT IN THE ARRAY. BY THE DIMENSION[A] OF AN ARRAY A WE SHALL MEAN THE

NUMBER OF ELEMENTS IN THE ARRAY. WE SHALL SAY THAT A NON-ARRAY NAME HAS DIMENSION[] = 1. THEREFORE EVERY NAME HAS A DIMENSION AS AN INHERITED ATTRIBUTE FROM ITS DECLARATION.

<ARRAY VARIABLE>A ::= <VARIABLE>V $[$ <SUBSCRIPT>B $]$

TYPE[A] = TYPE[V]
(TYPE[A] ≠ BOOLEAN)

SIZE[A(S)] = SIZE[V(S)]

ADDRESS[A(S)] = ADDRESS[V] + NSIZE[V] × VALUE[B(S)]
(0 ≤ VALUE[B(S)] < DIMENSION[V])
(NOTE THE USE OF THE INHERITED ATTRIBUTE NSIZE[V] WHICH IS NOT NECESSARILY THE SAME AS SIZE[V]).

VALUE[A(S)] = CONTENT[ADDRESS[A(S)], TYPE[A], SIZE[A(S)]]

SIDE[A(S)] = S" WHERE S"(IX1) = <INDETERMINABLE>, S"(X) = S(X) FOR X ≠ IX1

<BOOLEAN VARIABLE>B ::= <NAME>N

ADDRESS[B] = ADDRESS[N]

TYPE[B] = BOOLEAN

SIZE[B] = 1 BIT

VALUE[B] = VALUE[N]

<IDENTIFIER>I ::= <VARIABLE>V; J ::= <ARRAY VARIABLE>A
K ::= <BOOLEAN VARIABLE>B

ADDRESS[I] = ADDRESS[V]; ADDRESS[J] = ADDRESS[A];
ADDRESS[K] = ADDRESS[B]

TYPE[I] = TYPE[V]; TYPE[J] = TYPE[A]; TYPE[K] = BOOLEAN

SIZE[I(S)] = SIZE[V(S)]; SIZE[J(S)] = SIZE[A(S)];
SIZE[K] = 1 BIT

```
VALUE[I(S)] = VALUE[V(S)]; VALUE[J(S)] = VALUE[A(S)];
    VALUE[K(S)] = VALUE[B(S)]

SIDE[I(S)] = SIDE[V(S)]; SIDE[J(S)] = SIDE[A(S)];
    SIDE[K(S)] = S
```

ARITHEMTIC EXPRESSION

<ADDING OPERATOR>A ::= $+$; B ::= $-$

FUNCTION[A] = PLUS; FUNCTION[B] = MINUS

<TERM>T ::= <IDENTIFIER>I; U ::= <ARITHMETIC CONSTANT>A

VALUE[T(S)] = VALUE[I(S)]; VALUE[U] = VALUE[A]

TYPE[T] = TYPE[I]; TYPE[U] = TYPE[A]
(TYPE[I] = INTEGER OR SIGNED OR REAL OR NUMERIC)

SIDE[T(S)] = SIDE[I(S)]; SIDE[U(S)] = S

<FACTOR>T ::= <IDENTIFIER>I; U ::= <SIGNED ARITHMETIC CONSTANT>A

VALUE[T(S)] = VALUE[I(S)]; VALUE[U] = VALUE[A]

TYPE[T] = TYPE[I]; TYPE[U] = TYPE[A]
(TYPE[I] = INTEGER OR SIGNED OR REAL OR NUMERIC)

SIDE[T(S)] = SIDE[I(S)]; SIDE[U(S)] = S

THERE IS A SPECIAL 1 BIT REGISTER NOT IN THE USER-S
ADDRESS SPACE BUT ACCESSIBLE TO HIM CALLED "OVERFLOW".

<ADDING EXPRESSION>A ::= <TERM>T <ADDING OPERATOR>P <TERM>U

LET SMAX[A(S)] = MAXIMUM[SIZE[T(S)], SIZE[U(S)]]
LET FN[A(S)] = FUNCTION[P, VALUE[I(S)], VALUE[(S)]],
IE; THE USUAL ARITHMETIC RESULT OF THE EXPRESSION.

VALUE[A(S)] = IF SIZE[FN[A(S)]] > SMAX[A(S)] THEN NONE ELSE
FN[A(S)]

((TYPE[T] = REAL) IF AND ONLY IF (TYPE[U] = REAL))

TYPE[A] = IF TYPE[T] = REAL THEN REAL ELSE SIGNED

SIZE[A(S)] = SMAX[A(S)]

```
SIDE[A(S)] = S" WHERE S"(Z) = SIDE[U(SIDE[T(S)])], Z ≠
     OVERFLOW; S"(OVERFLOW) = IF VALUE[A(S)] = NONE THEN
     TRUE ELSE S(OVERFLOW)

CMP[A(S)] = IF SIDE[OVERFLOW] = TRUE THEN CMP[S] ELSE
     COMP[FN[A(S)]]
     (BY CMP[S] WE MEAN THE PREVIOUS VALUE OF CMP[]. NOTE:
     CMP[A(S)] DEPENDS ON THE RESULT OF THE ARITHMETIC
     OPERATION, NOT ON THE TYPES OF THE TERMS.)

<MULTIPLY EXPRESSION>M ::= <FACTOR>F $*$ <TERM>T

     VALUE[M(S)] = VALUE[F(S)] × VALUE[T(S)]

     SIZE[M(S)] = SIZE[F(S)] + SIZE[T(S)]

          ( (TYPE[T] = REAL) IF AND ONLY IF (TYPE[F] = REAL) )

     TYPE[M] = IF TYPE[F] = REAL THEN REAL ELSE IF (TYPE[F] =
          SIGNED OR TYPE[T] = SIGNED) THEN SIGNED  ELSE INTEGER

     SIDE[M(S)] = SIDE[T(SIDE[F(S)])]

     CMP[M(S)] = COMP[VALUE[M(S)]]

<DIVIDE EXPRESSION>D ::= <FACTOR>F $/$ <TERM>T;
     E ::= $[$ $REM$ $]$ <FACTOR>G $/$ <TERM>U

          LET FN[D(S)] = FUNCTION[/, VALUE[F(S)], VALUE[T(S)]]
          AND FN[E(S)] = FUNCTION[/, VALUE[G(S)], VALUE[U(S)]] IN
          THE USUAL ARITHMETIC SENSE.

          LET SZ[D(S)] = SIZE[F(S)] - SIZE[T(S)],
          AND SZ[E(S)] = SIZE[G(S)] - SIZE[U(S)]
          (SIZE[F(S)] > SIZE[T(S)]; SIZE[G(S)] > SIZE[U(S)])

     VALUE[D(S)] = IF (SIZE[FN[D(S)]] > SZ[D(S)]) THEN NONE ELSE
          FN[D(S)]

     VALUE[E(S)] = IF (SIZE[FN[E(S)]] > SZ[E(S)]) THEN NONE ELSE
          FN[E(S)]

     SIZE[D(S)] = SZ[D(S)]; SIZE[E(S)] = SZ[D(S)]
```

```
        ( (TYPE[F] = REAL) IF AND ONLY IF (TYPE[T] = REAL),
        TYPE[G] ≠ REAL, TYPE[U] ≠ REAL)

      TYPE[D] = IF TYPE[F] = REAL THEN REAL ELSE IF (TYPE[F] =
           SIGNED OR TYPE[T] = SIGNED) THEN SIGNED ELSE INTEGER

      TYPE[D] = IF (TYPE[G] = SIGNED OR TYPE[U] = SIGNED) THEN
           SIGNED ELSE INTEGER

      SIDE[D(S)] = S" WHERE S"(Z) = SIDE[T(SIDE[F(S)])], Z ≠
           OVERFLOW; S"(OVERFLOW) = IF VALUE[D(S)] = NONE THEN
           TRUE ELSE S(OVERFLOW)

      SIDE[E(S)] = S" WHERE S"(Z) = SIDE[U(SIDE[G(S)])], Z ≠
           OVERFLOW, Z ≠ G; S"(OVERFLOW) = IF VALUE[E(S)] = NONE
           THEN TRUE ELSE S(OVERFLOW); S"(G) = IF VALUE[E(S)] =
           NONE THEN S(G) ELSE
           VALUE[G(S)] = VALUE[E(S)] × VALUE[U(S)]  (IE: THE
           REMAINDER OF THE DIVISION)

      CMP[D(S)] = IF SIDE[OVERFLOW] = TRUE THEN CMP[S] ELSE
           COMP[FN[D(S)]]

      CMP[E(S)] = IF SIDE[OVERFLOW] = TRUE THEN CMP[S] ELSE
           COMP[FN[E(S)]]


<ARITHMETIC EXPRESSION>A ::= <ADDING EXPRESSION>E;
      B ::= <MULTIPLY EXPRESSION>M;  C ::= <DIVIDE EXPRESSION>V;
      D ::= <FACTOR>F

      VALUE[A(S)] = VALUE[E(S)]; VALUE[B(S)] = VALUE[M(S)];
           VALUE[C(S)] = VALUE[V(S)]; VALUE[D(S)] = VALUE[F(S)]

      SIZE[A(S)] = SIZE[E(S)]; SIZE[B(S)] = SIZE[M(S)];
           SIZE[C(S)] = SIZE[V(S)]; SIZE[D(S)] = SIZE[F(S)]

      SIDE[A(S)] = SIDE[E(S)]; SIDE[B(S)] = SIDE[M(S)];
           SIDE[C(S)] = SIDE[V(S)]; SIDE[D(S)] = SIDE[F(S)]

      TYPE[A] = TYPE[E]; TYPE[B] = TYPE[M]; TYPE[C] = TYPE[V];
           TYPE[D] = TYPE[F]

      CMP[A(S)] = CMP[E(S)]; CMP[B(S)] = CMP[M(S)];
           CMP[C(S)] = CMP[V(S)]; CMP[D(S)] = CMP[F(S)];
```

BOOLEAN EXPRESSION

<BOOLEAN TERM>T ::= <BOOLEAN CONSTANT>C; U ::= <BOOLEAN VARIABLE>V

VALUE[T] = VALUE[C]; VALUE[U(S)] = VALUE[V(S)]

<COMPARE OP>A ::= $=$; B ::= $ NEQ $; C ::= $<$;
D ::= $ LEQ $; E ::= $>$; F ::= $ GEQ $

VALUE[A] = :(EQUAL):; VALUE[B] = :(GREATER, LESS):;
VALUE[C] = :(LESS):; VALUE[D] = :(LESS, EQUAL):;
VALUE[E] = :(GREATER):; VALUE[F] = :(GREATER, EQUAL):

NOTE IN EACH CASE THE VALUE OF THE COMPARE OP IS A SET OF VALUES FOR THE COMPARISON INDICATOR WHICH ARE VALID FOR THAT COMPARE OP.

<ITEM>I ::= <IDENTIFIER>V; J ::= <SIGNED CONSTANT>C

VALUE[I(S)] = VALUE[V(S)]; VALUE[J] = VALUE[C]

TYPE[I] = TYPE[V]; TYPE[J] = TYPE[C]
(TYPE[I] ≠ BOOLEAN; TYPE[J] ≠ BOOLEAN)

SIDE[I(S)] = SIDE[V(S)]; SIDE[J(S)] = S

<COMPARISON>C ::= <ITEM>I <COMPARE OP>O <ITEM>J

COMPARISONS ARE DONE IN THE FOLLOWING MANNER: IF BOTH TYPE[I] AND TYPE[J] ARE ANY COMBINATION OF INTEGER, SIGNED, OR NUMERIC, THEN THE COMPARISON IS ON THE BASIS OF THE NUMERIC VALUE OF I AND J. IN ALL OTHER CASES, THE COMPARISON IS ON THE BASIS OF THE CORRESPONDING STRINGS. THE SHORTER STRING (IF ANY) IN THE NON-NUMERIC CASE IS PADDED WITH TRAILING BLANKS TO MAKE THE LENGTHS EQUAL. NUMBERS COMPARE HIGHER THAN LETTERS, LETTERS HIGHER THAN BLANKS.

VALUE[C(S)] = IF THE SPECIFIED COMPARISON IS TRUE THEN TRUE

ELSE FALSE

SIDE[C(S)] = SIDE[J(SIDE[I(S)])]

CMP[C(S)] = IF VALUE[I(S)] = VALUE[J(S)] THEN EQUAL ELSE IF VALUE[I(S)] < VALUE[J(S)] THEN LESS ELSE GREATER.

<BOOLEAN FACTOR>F ::= <COMPUTATIONAL HEX NUMBER>C;
G ::= <DISPLAY HEX NUMBER>D; H ::= <INTEGER>I

VALUE[F] = VALUE[C]; VALUE[G] = VALUE[D]; VALUE[H] = VALUE[I]

LENGTH[F] = LENGTH[C]; LENGTH[G] = LENGTH[D]; LENGTH[H] = LENGTH[I]
(LENGTH[F] ≤ 2; LENGTH[G] = 1; LENGTH[H] ≤ 2)

<BOOLEAN AND>A ::= <IDENTIFIER>I $ AND $ <BOOLEAN FACTOR>F

(TYPE[I] ≠ BOOLEAN)

LET MASK[I(S), F] BE THE STRING OF DIGITS OR CHARACTERS (DEPENDING ON TYPE[F]) THAT IS FORMED BY THE REPITION OF F A SUFFICIENT NUMBER OF TIMES SUCH THAT LENGTH[MASK[I(S), F]] ≥ LENGTH[I(S)], BOTH EXPRESSED IN THE SAME UNITS (DIGITS OR CHARACTERS).

LET BAND[I(S), F] BE THE RESULT OF A LOGICAL AND OF I AND MASK[I(S), F]; BAND[I(S), F] = I AND MASK[I(S), F]

VALUE[A(S)] = IF BAND[I(S), F] = 0 THEN FALSE ELSE TRUE

SIDE[A(S)] = SIDE[I(S)]

CMP[A(S)] = IF VALUE[A(S)] = TRUE THEN GREATER ELSE EQUAL

<BOOLEAN NOT>N ::= <IDENTIFIER>I $ NOT $ <BOOLEAN FACTOR>F

(TYPE[I] ≠ BOOLEAN)

VALUE[A(S)] = IF BAND[I(S), F] = MASK[I(S), F] THEN FALSE ELSE TRUE

```
SIDE[A(S)] = SIDE[1(S)]

CMP[A(S)] = IF VALUE[A(S)] = TRUE THEN GREATER ELSE EQUAL
```

THE <BOOLEAN AND> AND <BOOLEAN NOT> FEATURES ALLOW THE CHECKING OF INDIVIDUAL BITS. FOR EXAMPLE, IF SWITCH IS DECLARED TO BE OF TYPE INTEGER, LENGTH 1 (DIGIT)

```
     SWITCH AND 1
```

IS TRUE IF THE ONE BIT OF SWITCH IS SET AND FALSE IF THE ONE BIT IS NOT SET.

```
     SWITCH AND 3
```

IS TRUE IF EITHER THE ONE BIT OR THE TWO BIT IS SET AND FALSE IF NEITHER IS SET. THE NOT CHECKS FOR BITS NOT SET.

```
     SWITCH NOT 1
```

IS TRUE IF THE ONE BIT OF SWITCH IS NOT SET AND FALSE IF IT IS SET.

```
     SWITCH NOT 3
```

IS TRUE IF EITHER THE ONE BIT OR THE TWO BIT IS NOT SET AND FALSE IF BOTH ARE SET.

```
<SIMPLE BOOLEAN>A ::= <BOOLEAN VARIABLE>V; B ::= <BOOLEAN
     CONSTANT>W; C ::= $OVERFLOW$; D ::= <COMPARE OP>X; E ::=
     <COMPARISON>Y; F ::= <BOOLEAN AND>Z; G ::= <BOOLEAN NOT>U

     VALUE[A(S)] = VALUE[V(S)]; VALUE[B] = VALUE[W];
          VALUE[C(S)] = S(OVERFLOW);
          VALUE[D(S)] = IF CMP[S] <: VALUE[X(S)] THEN TRUE ELSE
          FALSE;
          VALUE[E(S)] = VALUE[Y(S)]; VALUE[F(S)] = VALUE[Z(S)];
          VALUE[G(S)] = VALUE[U(S)]

     SIDE[A(S)] = SIDE[V(S)]; SIDE[B] = S; SIDE[C(S)] = S"
          WHERE S"(N) = S(N), N ≠ OVERFLOW, S"(OVERFLOW) = FALSE;
          SIDE[D(S)] = S; SIDE[E(S)] = SIDE[Y(S)]; SIDE[F(S) =
          SIDE[Z(S)]; SIDE[G(S)] = SIDE[U(S)]

     CMP[A(S)] = IF VALUE[A(S)] = TRUE THEN GREATER ELSE EQUAL;
          CMP[B] = IF VALUE[B[ = TRUE THEN GREATER ELSE EQUAL;

          CMP[C(S)] = CMP[S]; CMP[D(S)] = CMP[S]; CMP[E(S)] =
          CMP[Y(S)]; CMP[F(S)] = CMP[Z(S)]; CMP[G(S)] =
          CMP[U(S)]

<BOOLEAN EXPRESSION>B ::= <SIMPLE BOOLEAN>T; C ::= $NOT $
```

<SIMPLE BOOLEAN>U

VALUE[B(S)] = VALUE[T(S)]; VALUE[C(S)] = IF VALUE[U(S)] =
TRUE THEN FALSE ELSE TRUE

SIDE[B(S)] = SIZE[T(S)]; SIDE[C(S)] = SIDE[U(S)]

CMP[B(S)] = CMP[T(S)]; CMP[C(S)] = CMP[U(S)]

LOGICAL EXPRESSION

<LOGICAL OP>L ::= $ AND $; M ::= $ OR $; N ::= $ NOT $

FUNCTION[L] = LOGICAL AND; FUNCTION[M] = LOGICAL OR;
FUNCTION[N] = LOGICAL EXCLUSIVE OR

<LOGICAL TERM>L ::= <SIMPLE CONSTANT>C; M ::= <IDENTIFIER>V

VALUE[L] = VALUE[C]; VALUE[M(S)] = VALUE[V(S)]

SIDE[L(S)] = S; SIDE[M(S)] = SIDE[V(S)]

TYPE[L] = TYPE[C]; TYPE[M] = TYPE[V]
(TYPE[C] = INTEGER, NUMERIC, ALPHA, OR COMP STRING;
TYPE[V] = INTEGER, NUMERIC, ALPHA, OR COMP STRING)

LENGTH[L] = LENGTH[C]; LENGTH[M(S)] = LENGTH[V(S)]

<LOGICAL EXPRESSION>E ::= <LOGICAL TERM>L <LOGICAL OP>O <LOGICAL TERM>M

VALUE[E(S)] = FUNCTION[O, VALUE[L(S)], VALUE[M(S)]]

(TYPE[L] = TYPE[M])

TYPE[E] = TYPE[L]

LENGTH[E(S)] = MAXIMUM[LENGTH[L(S)], LENGTH[M(S)]]

SIDE[E(S)] = SIDE[M(SIDE[L(S)])]

LET LEAST[X] BE THE VALUE OF THE ONE BIT OF THE LAST (RIGHT-MOST, LOW-ORDER) DIGIT OF X.

CMP[E(S)] = IF LEAST[VALUE[E(S)]] = 1 THEN GREATER ELSE EQUAL

STATEMENTS

IN THE SAME MANNER AS IN PROFESSOR MAUER-S REPORT. WE INTRODUCE ADDITIONAL ATTRIBUTES FOR STATEMENTS:

EFFECT[] -- A FUNCTION FROM THE CURRENT STATE OF SOME COMPUTATION TO THE STATE VECTOR AFTER THE COMPUTATION. NOTE: SIDE EFFECTS ARE INCLUDED IN THE EFFECT OF A STATEMENT.

INDEX[] -- THE STATEMENT INDEX OF A STATEMENT. DUE TO THE CHARACTERISTICS OF BPL, WE CHOOSE TO LET THE STATEMENT INDEX OF A STATEMENT BE THE COMPILER CHOSEN ADDRESS OF THE START OF THAT STATEMENT IN COMBINATION WITH THE BLOCK NUMBER OF THE BLOCK THE STATEMENT IS IN, BLOCK[]. THE BLOCK NUMBER IS DEFINED BELOW UNDER "SCOPE OF IDENTIFIERS". THE INDEX[] OF A STATMENT THEN DEFINES EACH STATEMENT (AT VARIOUS LEVELS) UNIQUELY, AND ALLOWS THE CONSTRUCTION OF A LABEL FUNCTION

LABEL[] -- A FUNCTION FROM A LABEL NAME AND BLOCK NUMBER TO A STATEMENT INDEX.

EXIT[] -- THE EXIT INDEX OF A STATEMENT IS THE STATEMENT INDEX OF THE NEXT STATEMENT TO BE EXECUTED. EXIT[] = NORMAL WILL BE TAKEN TO MEAN THE NEXT SUCCESSIVE STATEMENT.

IN ORDER TO DESCRIBE THE EFFECT OF A SECTION WHICH IS NOT A STRAIGHT-LINE SECTION (IE: HAS BRANCHES), WE INTRODUCE

STEP[] -- THE SINGLE STEP FUNCTION. STEP[] IS A FUNCTION OF THE PAIR (S, K) WHERE S IS THE CURRENT STATE VECTOR, AND K THE CURRENT STATEMENT INDEX TO THE PAIR (S", K"), THE NEW STATE VECTOR AND STATEMENT INDEX. THEN THE

GENEFFECT[] -- THE GENERALIZED EFFECT -- CAN BE DEFINED AS A FUNCTION OF THE SINGLE STEP FUCNTION.

INSTEAD OF HAVING A NUMBER OF STATEMENTS IN A SECTION ATTRIBUTE, WE INTRODUCE A

INDEXSET[] -- THE SET OF ALL STATEMENT INDICES OF THE ARGUMENT.

SEE REFERENCE 2 FOR ADDITIONAL INFORMATION ON THIS SUBJECT.

<SECTION>X ::= <STATEMENT>T;
Y ::= <SECTION>Z $;$ <STATEMENT>U

INDEXSET[X] = INDEXSET[T]; INDEXSET[Y] = INDEXSET[Z] .U. INDEXSET[U]

STEP[X]((S, INDEX[T])) = (EFFECT[T(S)], EXIT[T(S)])
STEP[Y]((S, K)) = STEP[Z]((S, K)) FOR K <: INDEXSET[Z],

```
        STEP[Y]((S, INDEX[U])) = (EFFECT[U(S)], EXIT[U(S)])

    GENEFFECT[X]((S, K)) = IF K <: INDEXSET[T] THEN
        GENEFFECT[X](STEP[X](S, K)) ELSE (S, K);
        GENEFFECT[Y]((S, K)) = IF K <: INDEXSET[Z] .U.
        INDEXSET[U] THEN GENEFFECT[Y](STEP[Y](S, K))
        ELSE (S, K)

    EFFECT[X(S)] = S" WHERE (S", K") =
        GENEFFECT[X]((S, INDEX[X])) FOR SOME K";
        EFFECT[Y(S)] = S" WHERE (S", K") =
        GENEFFECT[X]((S, INDEX[Y])) FOR SOME K"

    EXIT[XS)] = K" WHERE (S", K") =
        ENEFFECT[X]((S, INDEX[X])) FOR SOME K";
        EXIT[Y(S)] = K" WHERE (S", K") =
        GENEFFECT[Y]((S, INDEX[Y])) FOR SOME K"


<COMPOUND STATEMENT>X ::= $BEGIN $ <SECTION>T $ END$

    INDEXSET[X] = INDEXSET[T]

    GENEFFECT[X]((S, K)) = IF K <: INDEXSET[T] THEN
        GENEFFECT[X](STEP[T]((S, K))) ELSE (S, K)

    EFFECT[X(S)] = S" WHERE (S", K") =
        GENEFFECT[X]((S, INDEX[X])) FOR SOME K"

    EXIT[X(S)] = K" WHERE (S", K") =
        GENEFFECT[X]((S, INDEX[X])) FOR SOME K"


<BLOCK>X ::= $BEGIN $ <DECLARATIONS>D <SECTION>T $ END$

    INDEXSET[X] = INDEXSET[T]

    GENEFFECT[X]((S, K)) = IF K <: INDEXSET[T] THEN
        GENEFFECT[X](STEP[T]((S, K))) ELSE (S, K)

    EFFECT[X(S)] = S" WHERE (S", K") =
        GENEFFECT[X]((S, INDEX[X])) FOR SOME K"

    EXIT[X(S)] = K" WHERE (S", K") =
        GENEFFECT[X]((S, INDEX[X])) FOR SOME K"
```

<PROGRAM NAME> ::= <NAME> $:$ // <EMPTY>

<PROGRAM>P ::= <PROGRAM NAME> <BLOCK>B

INDEXSET[P] = INDEXSET[B]

EFFECT[P(S)] = EFFECT[B(S)]

EXIT[P(S)] = STOP WHERE STOP IS THE SPECIAL EXIT INDEX TO THE OPERATING SYSTEM (MCP) TO CAUSE THE PROGRAM TO TERMINATE.

A LABEL IS DEFINED BY ITS NAME AND THE BLOCK WITHIN WHICH IT IS DECLARED (WHICH MUST BE THE SAME AS THE BLOCK WITHIN WHICH IT APPEARS) AND ITS STATEMENT INDEX. DEFINE

LABEL[] -- THE LOCAL LABEL FUNCTION FROM A LABEL AND BLOCK NUMBER TO A STATEMENT INDEX.

<LABEL> ::= <NAME>

<STATEMENT>T ::= <BLOCK>B; U ::= COMPOUND STATEMENT>C; V ::= <SIMPLE STATEMENT>D; W ::= <LABEL>L $:$ <STATEMENT>E

INDEXSET[T] = INDEXSET[B]; INDEXSET[U] = INDEXSET[C]; INDEXSET[V] = INDEXSET[D]; INDEXSET[W] = INDEXSET[E]

EFFECT[T(S)] = EFFECT[B(S)]; EFFECT[U(S)] = EFFECT[C(S)]; EFFECT[V(S)] = EFFECT[D(S)]; EFFECT[W(S)] = EFFECT[E(S)]

EXIT[T(S)] = EXIT[B(S)]; EXIT[U(S)] = EXIT[C(S)]; EXIT[V(S)] = EXIT[D(S)]; EXIT[W(S)] = EFFECT[E(S)]

LABEL[T] = LABEL[B]; LABEL[U] = LABEL[C]; LABEL[V] = LABEL[D]; LABEL[W] = :((L. INDEX[E])): .U. LABEL[E]

<SIMPLE STATEMENT>P ::= <ASSIGNMENT STATEMENT>A;
Q ::= <CONDITIONAL STATEMENT>C;
R ::= <TRANSFER STATEMENT>B;
T ::= <PROCEDURE CALL>P;

```
U ::= <MCP REQUEST>M;
V ::= <CASE STATEMENT>D;
W ::= <LOOP STATEMENT>L;
X ::= <COMPARE STATEMENT>N;
Y ::= <CONVERT STATEMENT>E;
Z ::= <FIND STATEMENT>F

INDEXSET[]

EFFECT[P(S)] = EFFECT[A(S)]; EFFECT[Q(S)] = EFFECT[C(S)];
     EFFECT[R(S)] = EFFECT[B(S)];
     EFFECT[T(S)] = EFFECT[P(S)];
     EFFECT[U(S)] = EFFECT[M(S)];
     EFFECT[V(S)] = EFFECT[D(S)];
     EFFECT[W(S)] = EFFECT[L(S)];
     EFFECT[X(S)] = EFFECT[N(S)];
     EFFECT[Y(S)] = EFFECT[E(S)];
     EFFECT[Z(S)] = EFFECT[F(S)]

EXIT[P(S)] = EXIT[A(S)]; EXIT[Q(S)] = EXIT[C(S)];
     EXIT[R(S)] = EXIT[B(S)]; EXIT[T(S)] = EXIT[P(S)];
     EXIT[U(S)] = EXIT[M(S)]; EXIT[V(S)] = EXIT[D(S)];
     EXIT[W(S)] = EXIT[L(S)]; EXIT[X(S)] = EXIT[N(S)];
     EXIT[Y(S)] = EXIT[E(S)]; EXIT[Z(S)] = EXIT[F(S)]
```

ASSIGNMENT STATEMENT

IN ORDER TO EXACTLY DEFINE THE RESULT OF AN ASSIGNMENT STATEMENT UNDER THE MANY POSSIBLE COMBINATIONS OF TYPE AND SIZE IN AN ORDERLY AND EASILY UNDERSTOOD MANNER, DEFINE

ASSIGN[RS, RT; SS, ST, SV]

TO BE THE VALUE OF THE RECEIVING FIELD OF TYPE RT, SIZE RS WITH A SENDING FIELD OF TYPE ST, SIZE SS, AND VALUE SV. THIS IS DONE IN THE FOLLOWING TABLE.

/ / IS USED FOR ABSOLUTE VALUE.

ANY COMBINATIONS THAT DO NOT APPEAR ARE INVALID.

#SV IS USED TO MEAN THE NUMERIC SUBSET OF SV, IE: THE ODD ADDRESS DIGITS ONLY OF THE ALPHA SV. FOR EXAMPLE, THE CHARACTER A HAS AN EBCDIC REPRESENTATION AS C1, B AS C2, C AS C3. IF SV = "ABC", THEN #SV = 123.

@SV IS USED TO MEAN THE CHARACTER STRING FORMED BY APPENDING THE UNDIGIT F TO EACH DIGIT OF THE DIGIT/UNDIGIT STRING SV. NOTE F0 IS "0", ..., F9 IS "9" (HOWEVER, FA, FB, FC, FD, FE, AND FF ARE UNPRINTABLE CHARACTERS). THEREFORE, IF SV = 123, @SV = "123". IF SV = @ABC@, THEN @SV = %FAFBFC%.

"NONE" MEANS NO VALUE IS FORMED. USUALLY OVERFLOW WILL BE SET IN THESE CASES.

THE COLUMN HEADER "RS ? SS" WILL CONTAIN COMPARISON INDICATORS TO INDICATE THE RELATIVE SIZE OF THE RECEIVING AND SENDING FIELDS.

THE ST AND RT FIELDS STAY THE SAME DOWN A COLUMN UNTIL CHANGED.

RT	ST	RS ? SS	ASSIGN[RS, RT; SS, ST, SV]
INTEGER	INTEGER	=	SV
		>	REPEAT[0,RS-SS] : SV
		<	NONE
	COMP STR	=	SV
		>	REPEAT[0,RS-SS] : SV
		<	NONE
	NUMERIC	=	#SV
		>	REPEAT[0,RS-SS] : #SV
		<	NONE
	SIGNED	=	/SV/
		>	REPEAT[0,RS-SS] : /SV/
		<	NONE
	ALPHA	=	#SV
		>	#SV : REPEAT[0,RS-SS]
		<	NONE
	BOOLEAN	RS = 1	IF SV = TRUE THEN 1 ELSE 0
		RS > 1	IF SV = TRUE THEN REPEAT[0,RS-1] : 1 ELSE REPEAT[0,RS]
COMP STR	INTEGER	=	SV
		>	REPEAT[0,RS-SS] : SV
		<	NONE
	COMP STR	=	SV
		>	REPEAT[0,RS-SS] : SV
		<	NONE
	NUMERIC	=	#SV
		>	REPEAT[0,RS-SS] : #SV
		<	NONE
	SIGNED	=	/SV/
		>	REPEAT[0,RS-SS] : /SV/
		<	NONE
	ALPHA	=	#SV
		>	#SV : REPEAT[0,RS-SS]
		<	NONE

RT	ST	RS ? SS	ASSIGN[RS, RT; SS, ST, SV]
	BOULEAN	RS = 1	IF SV = TRUE THEN 1 ELSE 0
		RS > 1	IF SV = TRUE THEN REPEAT[0,RS-1] : 1 ELSE REPEAT[0,RS]
NUMERIC	INTEGER	=	@SV
		>	REPEAT[0,RS-SS] : @SV
		<	NONE
	COMP STR	=	@SV
		>	REPEAT[0,RS-SS] : @SV
		<	NONE
	NUMERIC	=	SV
		>	REPEAT[0,RS-SS] : SV
		<	NONE
	SIGNED	=	/SV/
		>	REPEAT[0,RS-SS] : /SV/
		<	NONE
	ALPHA	=	SV
		>	SV : REPEAT[$ $,RS-SS]
		<	NONE
ALPHA	INTEGER	=	@SV
		>	REPEAT[0,RS-SS] : @SV
		<	NONE
	COMP STR	=	@SV
		>	REPEAT[0,RS-SS] : @SV
		<	NONE
	NUMERIC	=	SV
		>	SV : REPEAT[$ $,RS-SS]
		<	NONE
	ALPHA	=	SV
		>	SV : REPEAT[$ $,RS-SS]
		<	NONE

RT	ST	RS ? SS	ASSIGN[RS, RT; SS, ST, SV]
SIGNED	INTEGER	=	SV
		>	REPEAT[0,RS-SS] : SV
		<	NONE
	COMP STR	=	SV
		>	REPEAT[0,RS-SS] : SV
		<	NONE
	NUMERIC	=	#SV
		>	REPEAT[0,RS-SS] : #SV
		<	NONE
	SIGNED	=	SV
		>	REPEAT[0,RS-SS] : SV
		<	NONE
	BOOLEAN	RS = 1	IF SV = TRUE THEN 1 ELSE 0
		RS > 1	IF SV = TRUE THEN REPEAT[0,RS-1] : 1 ELSE REPEAT[0,RS]
BOOLEAN	BOOLEAN		SV

```
AA ::= <IDENTIFIER>A $←$ <CONSTANT>K
     BB ::= <IDENTIFIER>B $←$ <IDENTIFIER>L
     CC ::= <IDENTIFIER>C $←$ <ARITHMETIC EXPRESSION>M
     DD ::= <IDENTIFIER>D $←$ <LOGICAL EXPRESSION>N
     EE ::= <IDENTIFIER>E $←$ $[$ $XCH$ $]$ <IDENTIFIER>P
     FF ::= <IDENTIFIER>F $←$ $[$ $XCH$ $]$ <IDENTIFIER>Q
            <BLANKS> <IDENTIFIER>R
     GG ::= <IDENTIFIER>G $←$ $[$ $WDS$ $]$ <IDENTIFIER>T
     HH ::= <IDENTIFIER>H $←$ $[$ $MVC$ $]$ <IDENTIFIER>U
     II ::= <IDENTIFIER>I $←$ $[$ $CHR$ $]$ <IDENTIFIER>V
     JJ ::= <IDENTIFIER>J $←$ $[$ $TGL$ $]$ <IDENTIFIER>W

     EFFECT[AA(S)] = S" WHERE
          S"(A(S))] = ASSIGN[LENGTH[A(S)], TYPE[A]; LENGTH[K],
               TYPE[K], VALUE[K],
          S"(OVERFLOW) = IF S"(A(S)) = NONE THEN TRUE ELSE
               S(OVERFLOW),
          S"(X) = SIDE[A(S(S))] OTHERWISE

     EFFECT[BB(S)] = S" WHERE
```

```
      S"(B(S))] = ASSIGN[LENGTH[B(S)], TYPE[B]; LENGTH[L(S)],
            TYPE[L], VALUE[L(SIDE[B(S)]))],
      S"(OVERFLOW) = IF S"(B(S)) = NONE THEN TRUE ELSE
            S(OVERFLOW),
      S"(X) = SIDE[L(SIDE[B(S(X))])] OTHERWISE

EFFECT[CC(S)] = S" WHERE
      S"(C(S))] = ASSIGN[LENGTH[C(S)], TYPE[C]; LENGTH[M(S)],
            TYPE[M], VALUE[M(SIDE[C(S)])]],
      S"(OVERFLOW) = IF S"(C(S)) = NONE THEN TRUE ELSE
            S(OVERFLOW),
      S"(X) = SIDE[M(SIDE[C(S(X))])] OTHERWISE

EFFECT[DD(S)] = S" WHERE
      S"(D(S))] = ASSIGN[LENGTH[D(S)], TYPE[D]; LENGTH[N(S)],
            TYPE[N], VALUE[N(SIDE[D(S)])]],
      S"(OVERFLOW) = IF S"(D(S)) = NONE THEN TRUE ELSE
            S(OVERFLOW),
      S"(X) = SIDE[N(SIDE[D(S(X))])] OTHERWISE

EFFECT[EE(S)] = S" WHERE
      S"(E(S)) = VALUE[P(SIDE[E(S)])],
      S"(P(SIDE[E(S)])) = VALUE[E(S)],
      S"(S) = SIDE[P(SIDE[E(S(X))])] OTHERWISE

EFFECT[FF(S)] = S" WHERE
      S"(F(S)) = VALUE[(SIDE[F(S)])],
      S"(Q(SIDE[E(S)])) = VALUE[R(SIDE[Q(SIDE[E(S)])])],
      S"(R(SIDE[Q(SIDE[E(S)])])) = VALUE[F(S)],
      S"(X) = SIDE[R(SDIE[Q(SIDE[F(S(X))])])] OTHERWISE

EFFECT[GG(S)] = S" WHERE
      S"(G(S)) = VALUE[T(SIDE[G(S)])],
      S"(X) = SIDE[T(SIDE[G(S(X))])] OTHERWISE

EFFECT[HH(S)] = S" WHERE
      S"(H(S)) = VALUE[U(SIDE[H(S)])],
      S"(U(SIDE[H(S)])) = IF TYPE[U] = INTEGER OR NUMERIC
            THEN ASSIGN[LENGTH[U(S)], ALPHA; LENGTH[U(S)],
            ALPHA, REPEAT[%00%, LENGTH[U(S)]]] ELSE
            ASSIGN[LENGTH[U(S)], INTEGER; 1, INTEGER, 0],
      S"(X) = SIDE[U(SIDE[H(S(X))])] OTHERWISE

EFFECT[II(S)] = S" WHERE
      S"(I(S))] = ASSIGN[LENGTH[I(S)], TYPE[I]; LENGTH[V(S)],
            TYPE[V], VALUE[V(SIDE[I(S)])]],
```

```
     S"(OVERFLOW) = IF S"(I(S)) = NONE THEN TRUE ELSE
          S(OVERFLOW),
     S"(X) = SIDE[V(SIDE[I(S(X))])] OTHERWISE

EFFECT[JJ(S)] = S" WHERE
     S"(J(S)] = ASSIGN[LENGTH[J(S)], TYPE[J]; LENGTH[W(S)],
          TYPE[W], VALUE[W(SIDE[J(S)])]],
     S"(OVERFLOW) = IF S"(J(S)) = NONE THEN TRUE ELSE
          S(OVERFLOW),
     S"(X) = SIDE[W(SIDE←J(S(X))])] OTHERWISE

EXIT[AA] = NORMAL; EXIT[BB] = NORMAL; EXIT[CC] = NORMAL;
           EXIT[DD] = NORMAL; EXIT[EE] = NORMAL;
           EXIT[FF] = NORMAL; EXIT[GG] = NORMAL;
           EXIT[HH] = NORMAL; EXIT[II] = NORMAL;
           EXIT[JJ] = NORMAL

(TYPE[A] AND TYPE[K] VALID TYPE PAIR ACCORDING TO ABOVE
     TABLE; LENGTH[A(S)] < 100;
TYPE[B] AND TYPE[L] VALID TYPE PAIR ACCORDING TO ABOVE
     TABLE; IF LENGTH[B(S)] ≥ 100 THEN (LENGTH[B(S)] =
     LENGTH[L(SIDE[B(S)])] AND TYPE[B] = TYPE[L] AND
     ADDRESS[B(S)] = 0 MOD 4 AND ADDRESS[L(SIDE[B(S)])] = 0
     MOD 4) ELSE LENGTH[L(S)] < 100;
TYPE[C] = INTEGER OR NUMERIC OR SIGNED; LENGTH[C(S)] < 100;
TYPE[D] ≠ (SIGNED OR REAL OR BOOLEAN); LENGTH[D(S)] < 100;
TYPE[E] = TYPE[P]; LENGTH[E(S)] = LENGTH[P(SIDE[E(S)])];
TYPE[F] = TYPE[Q] = TYPE[R]; LENGTH[F(S)] =
     LENGTH[Q(SIDE[F(S)])] = LENGTH[R(SIDE[Q(SIDE[F(S)])])];
TYPE[G] ≠ (SIGNED OR REAL OR BOOLEAN); TYPE[T] ≠ (SIGNED OR
     REAL OR BOOLEAN); LENGTH[G(S)] =
     LENGTH[T(SIDE[G(S)])]; ADDRESS[G(S)] = 0 MOD 4;
     ADDRESS[T(SIDE[G(S)])] = 0 MOD 4;
TYPE[H] ≠ (SIGNED OR REAL OR BOOLEAN); TYPE[U] ≠ (SIGNED OR
     REAL OR BOOLEAN); IF TYPE[H] = ALPHA OR NUMERIC THEN
     TYPE[U] = ALPHA OR NUMERIC; LENGTH[H(S)] =
     LENGTH[U(SIDE[H(S)])]; ADDRESS[H(S)] = 0 MOD 4;
     ADDRESS[U(SIDE[H(S)])] = 0 MOD 4;
TYPE[I] AND TYPE[V] VALID TYPE PAIR ACCORDING TO ABOVE
     TABLE; TYPE[I] ≠ BOOLEAN;
TYPE[J] AND TYPE[W] VALID TYPE PAIR ACCORDING TO ABOVE
     TABLE; TYPE[J] ≠ BOOLEAN)

CMP[AA(S)] = IF TYPE[A] ≠ (BOOLEAN OR ALPHA) AND TYPE[K] ≠
     (BOOLEAN OR ALPHA) AND LENGTH[A(S)] ≠
     LENGTH[K] THEN COMP[VALUE[K]] ELSE <INDETERMINABLE>;
```

```
CMP[BB(S)] = IF TYPE[B] ≠ (BOOLEAN OR ALPHA) AND TYPE[L] ≠
     (BOOLEAN OR ALPHA) AND LENGTH[A(S)] ≠
     LENGTH[L(SIDE[B(S)])] THEN
     COMP[VALUE[L(SIDE[B(S)])]] ELSE <INDETERMINABLE>;

CMP[CC] = CMP[M(SIDE[C(S)])];

CMP[DD] = CMP[N(SIDE[D(S)])];

CMP[EE] = CMP[S];  CMP[FF] = CMP[S];

CMP[GG] = CMP[S];  CMP[HH] = CMP[S];

CMP[II] = <INDETERMINABLE>;

CMP[JJ] = IF TYPE[W] = (ALPHA OR NUMERIC) THEN
     COMP[#VALUE[W(SIDE[J(S)])]] ELSE
     COMP[VALUE[W(SIDE[J(S)])]]
```

IN FORM BB (ONLY) THE FOLLOWING RESERVED WORDS MAY BE USED:

RESERVED WORD (L)	LENGTH [B(S)]	TYPE [B]	VALUE[B(S)] (RESULT) AND COMMENTS
DATE	0	INTEGER	RUN TIME DATE IN FORM MMDDYY.
JDATE	3	INTEGER	RUN TIME DATE IN FORM YYDDD.
MEMORY	0	INTEGER	RUN TIME CORE ASSIGNED TO PROGRAM (IN FORM CCCXXX WHERE CCC IS NUMBER OF THOUSANDS OF DIGITS ASSIGNED, XXX INDETERMINABLE).
REMAINDER	?	?	REMAINDER OF THE LAST DIVIDE REQUEST (UNLESS [REM] WAS SPECIFIED ON THE DIVIDE). SIZE AND TYPE DEPEND ON FORM OF THE ARITHMETIC INSTRUCTION INVOLVED. IN GENERAL B SHOULD BE THE SAME SIZE AND OF THE SAME TYPE AS THE DIVIDEND.
ROUTINETYPE	4	INTEGER	INFORMATION NEEDED WHEN A USER CHOOSES TO PROCESS HIS OWN FILE LABELS. USE AND CONTENTS BEYOND THE SCOPE OF THIS PAPER.
[SEGDICT]	6	INTEGER	ADDRESS OF SEGMENT DICTIONARY. USE BEYOND THE SCOPE OF THIS PAPER.
TIME	10	INTEGER	TIME OF DAY IN MILLESECONDS.
TIME60	10	INTEGER	TIME OF DAY IN HOURS, MINUTES, SECONDS, AND 60THS OF SECONDS IN FORM OOHHMMSSXX.

CONDITIONAL STATEMENT

```
<CONDITIONAL STATEMENT>X ::= $IF $ <BOOLEAN EXPRESSION>B $ THEN $
        <STATEMENT>T;
     Y ::= $IF $ <BOOLEAN EXPRESSION>C $ THEN $ <STATEMENT>U
        $ ELSE $ <STATEMENT>V

     EFFECT[X(S)] = IF VALUE[B(S)] = TRUE THEN
        EFFECT[T(SIDE[B(S)])] ELSE S;

     EFFECT[Y(S)] = IF VALUE[C(S)] = TRUE THEN
        EFFECT[U(SIDE[C(S)])] ELSE EFFECT[V(SIDE[C(S)])]

     EXIT[X(S)] = IF VALUE[B(S)] = TRUE THEN EXIT[T(SIDE[B(S)])]
        ELSE NORMAL;

     EXIT[Y(S)] = IF VALUE[C(S)] = TRUE THEN EXIT[U(SIDE[C(S)])]
        ELSE EXIT[V(SIDE[C(S)])]

     CMP[X(S)] = IF VALUE[B(S)] = TRUE THEN CMP[T(SIDE[B(S)])]
        ELSE CMP[B(S)];

     CMP[Y(S)] = IF VALUE[C(S)] = TRUE THEN CMP[U(SIDE[C(S)])]
        ELSE CMP[V(SIDE[C(S)])]
```

TRANSFER STATEMENT

<TRANSFER COMMAND> ::= $GO $ // <TRANSFER COMMAND> $TO $

(YES, " GO TO TO " IS VALID.)

<TRANSFER STATEMENT>X ::= <TRANSFER COMMAND> <LABEL>L;
Y ::= <TRANSFER COMMAND> <VARIABLE>I

EFFECT[X(S)] = S; EFFECT[Y(S)] = SIDE[I(S)]

EXIT[X(S)] = LABEL[L]; EXIT[Y(S)] = ADDRESS[I(S)]

CMP[X(S)] = CMP[S]; CMP[Y(S)] = IF SIDE[I(S)] = S THEN CMP[S] ELSE CMP[SIDE[I(S)]]

(USUALLY THE SECOND FORM IS USED IN THE CASE OF INDIRECTION, IE: I HAS A <TYPE MODIFIER> OF "IA". HOWEVER, IT IS VALID TO TRANSFER TO A DATA NAME DIRECTLY.)

PROCEDURE CALL

<PROCEDURE CALL>P ::= <PROCEDURE NAME>N <PROCEDURE ARGUMENT>A

EFFECT[P(S)] = EFFECT[PROCEDURE N(S)]

EXIT[P(S)] = EXIT[N(S)]

<PROCEDURE NAME> ::= <NAME>

<ARGUMENT> ::= <CONSTANT> // <IDENTIFIER> // <LABEL> //
<PROCEDURE NAME>

<ARGUMENT LIST>A ::= <ARGUMENT>X;
B ::= <ARGUMENT LIST>L $,$ <ARGUMENT>Y

NUMARGS[A] = 1; NUMARGS[B] = NUMARGS[L] + 1
(NUMARGS[L] ≤ 9)

ARGNUM[X] = 1; ARGNUM[Y] = NUMARGS[L] + 1

<PROCEDURE ARGUMENT>A ::= <EMPTY>;
B ::= $($ <ARGUMENT LIST>L $)$

NUMARGS[A] = 0; NUMARGS[B] = NUMARGS[L]

THE NUMBER OF ARGUMENTS MUST BE THE SAME AS THE NUMBER OF FORMAL PARAMETERS IN THE PROCEDURE DECLARATION FOR <PROCEDURE NAME>. EACH ARGUMENT MUST BE OF THE SAME TYPE AS THE CORRESPONDING FORMAL PARAMETER. IF THE FORMAL PARAMETER IS A CALL-BY-NAME PARAMETER THE LENGTH OF THE ARGUMENT MUST BE THE SAME AS THE LENGTH OF THE FORMAL PARAMETER. ARGUMENTS OF THE FORM <LABEL> OR <PROCEDURE NAME> MUST CORRESPOND TO CALL-BY-NAME PARAMETERS.
PROCEDURES MAY BE CALLED RECURSIVELY AND/OR NESTED IN ANY FASHION. SEE "PROCEDURE DECLARATIONS" BELOW.

MCP REQUEST

THIS SECTION DEALS WITH ALL STATEMENTS THAT INVOLVE PROCESSING BEING DONE FOR THE USER BY THE MCP (MASTER CONTROL PROGRAM). MOST MCP REQUESTS DEAL WITH I/O OPERATIONS. EVERY I/O OPERATION MUST BE DONE TO A FILE. A FILE IS IDENTIFIED BY A

<FILE NAME> ::= <NAME>.

FILE NAMES MUST BE UNIQUE IN THE FIRST 6 CHARACTERS WITHIN A PROGRAM. WITHOUT GETTING INTO VERY MUCH DETAIL OR DISCUSSING VARIOUS SPECIAL PROCESSING WHICH CAN BE DONE, EACH FILE HAS A RECORD AREA TO HOLD THE CURRENT RECORD BEING PROCESSED. THIS RECORD AREA WILL BE NOTED BY

RECORD[F]

WHERE F IS THE FILE NAME. THE SPECIAL WORDS INPUT[F] AND OUTPUT[F] WILL BE USED TO INDICATE THE FACT THAT A RECORD HAS BEEN READ OR A RECORD WRITTEN.

THEREFORE BY S WE MEAN THE STATE VECTOR NOT INCLUDING THE CONTENTS OF ANY FILES.

NO ATTEMPT WILL BE MADE TO COMPLETELY DEFINE THE EFFECT TO A FILE DUE TO BLOCKING, OR WHAT HAPPENS WITH RANDOM DISK FILES, ETC.

SINCE ONE DISK FILE MAY BE SIMULTANEOUSLY ACCESSED BY SEVERAL PROGRAMS IN EACH OF SEVERAL SEPERATE PROCESSORS (SHARED FILE CONCEPT, NOT MULTIPROCESSING), A FACILITY HAS BEEN IMPLEMENTED TO PREVENT OTHER PROGRAMS FROM ACCESSING A FILE OR PART OF A FILE DURING CRITICAL PATH PROCESSING. THIS IS CALLED "LOCK", AND CAN BE DONE ON A FILE OR RECORD BASIS.

THIS DISCUSION ASSUMES THAT THE USER IS NOT "FIB-FIDDLING", MODIFYING THE CONTENTS OF HIS FILE INFORMATION BLOCK DURING HIS RUN. SOME USEFUL ROUTINES CAN BE DEVELOPED USING THESE METHODS, BUT THEY DO NOT LEND THEMSELVES TO A RIGOROUS DISCUSION ON THIS LEVEL.

BY "EOF" WE MEAN THE END-OF-FILE CONDITION HAS BEEN SENSED ON THE FILE IN QUESTION. IT CAN ALSO MEAN THAT THE KEY FOR A RANDOM DISK FILE IS INVALID.

BY "ABORT" WE MEAN THAT THE ACTION IS INVALID AND THE PROGRAM IS TO BE TERMINATED BY THE MCP.

THE CONCEPT OF A VARIABLE WHOSE ATTRIBUTES ARE FIXED WILL PROVE USEFUL:

<CLEAN VARIABLE>A ::= <VARIABLE>V

(V CAN NOT HAVE A <TYPE MODIFIER> OF "IA"; V CAN NOT BE INDEXED; V CAN NOT HAVE AN INDIRECT FIELD LENGTH APPLIED)

TYPE[A] = TYPE[V]

LENGTH[A] = LENGTH[V]

ADDRESS[A] = ADDRESS[V]

<ACCEPT STATEMENT>A ::= $ACCEPT $ <CLEAN VARIABLE>V

(TYPE[V] = ALPHA)

EFFECT[A(S)] = S" WHERE S"(V) = <OPERATOR MESSAGE>, S"(X) = S OTHERWISE

EXIT[A(S)] = NORMAL

THIS STATEMENT ALLOWS THE PROGRAM TO RECEIVE DATA FROM THE SYSTEM OPERATOR. NO FILE IS INVOLVED. THE DATA IS PLACED IN V JUST AS THE OPERATOR TYPED IT IN, STARTING WITH THE FIRST CHARACTER OF V. IF THE OPERATOR TYPED IN FEWER CHARACTERS THAN LENGTH[V] THEN THE DATA IS FOLLOWED BY AN ETx (%03%). IF THE OPERATOR TYPED MORE THAN LENGTH[V] CHARACTERS, ONLY THE FIRST LENGTH[V] ARE PLACED IN V.

<CLOSE STATEMENT>C ::= $CLOSE $ <REEL> <FILE NAME> <CLOSE TYPE> <CRUNCH>

EXIT[C(S)] = NORMAL

THE CLOSE STATEMENT CLOSES THE SPECIFIED FILE ACCORDING TO <CLOSE TYPE>. THE FILE MUST HAVE BEEN OPEN PRIOR TO THE CLOSE.

<REEL> ::= <EMPTY> // $REEL $
CAN BE "REEL" ONLY FOR TAPE FILES TO INDICATE THAT THE CURRENT REEL (OF A MULTI-REEL FILE) IS TO BE CLOSED. (THE FILE REMAINS OPEN). THE END OF REEL ACTION IS PERFORMED AUTOMATICALLY BY THE MCP WHEN PHYSICAL END OF

REEL IS REACHED.

<CLOSE TYPE> ::= <EMPTY> // $ PURGE$ // $ RELEASE$ // $ REWIND$ // $ NO REWIND$ // $ NO REWIND RELEASE$ // $ REMOVE$ // $ LOCK$ // $ NO DISCONNECT$

INDICATES TYPE OF CLOSE. "NO DISCONNECT" IS VALID FOR DATA COMM FILES ONLY.

<CRUNCH> ::= <EMPTY> // $ CRUNCH$
"CRUNCH" ONLY VALID ON DISK FILES.

<DATA COMM I/O STATEMENT> -- BEYOND THE SCOPE OF THIS PAPER.

<DISPLAY STATEMENT>A ::= $DISPLAY $ <CLEAN VARIABLE>V;
B ::= $DISPLAY $ <ALPHA CONSTANT>C

(LENGTH[V] ≤ 60; TYPE[V] = ALPHA; LENGTH[C] ≤ 60)

EFFECT[A(S)] = S; EFFECT[B(S)] = S

EXIT[A(S)] = NORMAL; EXIT[B(S)] = NORMAL

THE VALUE[V(S)] OR VALUE[C] IS DISPLAYED TO THE SYSTEM OPERATOR.

<DOZE STATEMENT>A ::= $DOZE $ <INTEGER>I;
B ::= $DOZE $ <CLEAN VARIABLE>V

EFFECT[A(S)] = S; EFFECT[B(S)] = S

EXIT[A(S)] = NORMAL; EXIT[B(S)] = NORMAL

(VALUE[I] < 100000; TYPE[V] = INTEGER; LENGTH[V] ≤ 5)

THE DOZE STATEMENT SUSPENDS THE PROGRAM FOR THE SPECIFIED NUMBER OF SECONDS (EITHER VALUE[I] OR VALUE[V(S)] SECONDS). AT THE END OF THAT TIME, THE PROGRAM IS REINSTATED.

<DUMP STATEMENT>A ::= $DUMP$

EFFECT[A(S)] = S

EXIT[A(S)] = NORMAL

THE DUMP STATEMENT CAUSES A COMPLETE CORE IMAGE DUMP OF ALL THE PROGRAM-S MEMORY AND SPECIAL REGISTERS TO BE TAKEN. THE DUMP CAN BE PRINTED FOR ANALYSIS.

<FILL STATEMENT>A ::= $FILL$ $ IN $ <CLEAN VARIABLE>V <WHOM>P
B ::= $FILL$ $ IN $ <CLEAN VARIABLE>W <WHOM>Q
$[$ <LABEL>L $]$;
C ::= $FILL$ $ OUT $ <CLEAN VARIABLE>Y <WHOM>R;
D ::= $FILL$ $ OUT $ <CLEAN VARIABLE>Z <WHOM>T
$[$ <LABEL>M $]$

EFFECT[A(S)] = S" WHERE S"(V) = CORE-TO-CORE MESSAGE, S"(X) = S OTHERWISE;
EFFECT[B(S)] = S" WHERE S"(W) = IF SENDER READY THEN CORE-TO-CORE MESSAGE ELSE S(W), S"(X) = S OTHERWISE
EFFECT[C(S)] = S; EFFECT[D(S)] = S

EXIT[A(S)] = NORMAL; EXIT[C(S)] = NORMAL;
EXIT[B(S)] = IF SENDER NOT READY THEN LABEL[L] ELSE NORMAL;
EXIT[D(S)] = IF RECEIVER NOT READY THEN LABEL[M] ELSE NORMAL

<WHOM>A ::= <ALPHA CONSTANT>L; B ::= <CLEAN VARIABLE>V

(LENGTH[L] ≤ 6; LENGTH[V] = 6; TYPE[V] = ALPHA)

THE FILL STATEMENT IS ONE FORM OF INTER-PROGRAM COMMUNICATION. IN THE FIRST TWO FORMS, THIS PROGRAM IS WAITING FOR A MESSAGE FROM PROGRAM <WHOM>. IN THE LAST TWO CASES, THIS PROGRAM IS WAITING TO SEND A MESSAGE TO PROGRAM <WHOM>. BOTH PROGRAMS MUST BE READY TO SEND AND RECEIVE BEFORE THE TRANSFER TAKES PLACE (IN CASES A AND C THE PROGRAM WILL BE SUSPENDED UNTIL THE OTHER PROGRAM IS READY. IN CASES B AND D, IF THE OTHER PROGRAM IS NOT READY, CONTROL WILL BE TRANSFERED TO <LABEL>.) IF VALUE[<WHOM>] = " " THEN THE TRANSFER WILL TAKE PLACE WITH ANY MATCHING PROGRAM. SEE THE STUQUE STATEMENT FOR ANOTHER TYPE OF INTER-PROGRAM COMMUNICATION WHICH DOES NOT REQUIRE BOTH PROGRAMS TO BE READY SIMULTANEOUSLY.

<MICR STATEMENT> -- BEYOND THE SCOPE OF THIS PAPER.

<OPEN STATEMENT>A ::= $OPEN$ <OPEN MODE> <FILE NAME> <OPEN TYPE>

EFFECT[A(S)] = S

EXIT[A(S)] = NORMAL

OPENS A FILE. FILE MUST HAVE BEEN CLOSED BEFORE THE OPEN REQUEST. NO I/O CAN BE DONE TO A FILE BEFORE IT IS OPENED.

<OPEN MODE> ::= $ OUT $ // $ IN $ // $ IO $ // $ OI $
INDICATES TYPE OF ACTION TO BE DONE TO THE FILE: OUTPUT ONLY (NEW FILE), INPUT ONLY (EXISTING FILE), INPUT AND OUTPUT (EXISTING FILE), OUTPUT AND INPUT (NEW FILE) RESPECTIVELY. "IO" AND "OI" VALID FOR DISK FILES ONLY.

<OPEN TYPE> ::= <EMPTY> // $ LOCK$ // $ LOCK ACCESS$ //
$ REVERSE$ // $ NO REWIND$
"NO REWIND" VALID FOR "OUT" OR "OI" ONLY; "LOCK" OR "LOCK ACCESS" VALID FOR "IO" OR "OI" DISK FILES ONLY.

<READ STATEMENT>A ::= $READ $<FILE NAME>F <LOCK>;
B ::= $READ $ <FILE NAME>G <LOCK> $[$ <LABEL>L $]$

EFFECT[A(S)] = IF EOF THEN S ELSE S" WHERE S"(RECORD[F]) = INPUT[F], S"(X) = S OTHERWISE
EFFECT[B(S)] = IF EOF THEN S ELSE S" WHERE S"(RECORD[G]) = INPUT[G], S"(X) = S OTHERWISE

EXIT[A(S)] = IF EOF THEN ABORT ELSE NORMAL;
EXIT[B(S)] = IF EOF THEN LABEL[L] ELSE NORMAL

<LOCK>A ::= <BLANKS>; B ::= $ WITH $ $LOCK $;
C ::= $ LOCK $

FORMS B AND C ARE VALID FOR DISK FILES ONLY.

<SEEK STATEMENT>A ::= $SEEK $ <FILE NAME> <LOCK>

VALID FOR RANDOM DISK FILES ONLY. THE EXACT EFFECT OF

A SEEK STATEMENT IS BEYOND THE SCOPE OF THIS PAPER. THE FILE MUST BE OPEN PRIOR TO THE SEEK STATEMENT BEING EXECUTED.

EXIT[A(S)] = IF EOF THEN ABORT ELSE NORMAL.

<SORT STATEMENT> -- BEYOND THE SCOPE OF THIS PAPER.

<SPOMESSAGE STATEMENT>A ::= $SPOMESSAGE $ <CLEAN VARIABLE>V <BLANKS> <CLEAN VARIABLE>W; B ::= $SPOMESSAGE $ <ALPHA CONSTANT>C <BLANKS> <CLEAN VARIABLE>Z

(ADDRESS[W] = 0 MOD 2, ADDRESS[Z] = 0 MOD 2; TYPE[W] = ALPHA)

EFFECT[A(S)] = S" WHERE S"(W) = <SYSTEM RESPONSE TO VALUE[V(S)]>, S"(X) = S OTHERWISE;
EFFECT[B(S)] = S" WHERE S"(Z) = <SYSTEM RESPONSE TO VALUE[C]>, S"(X) = S OTHERWISE

EXIT[A(S)] = NORMAL; EXIT[B(S)] = NORMAL

V OR C MAY CONTAIN ANY VALID SYSTEM OPERATOR REQUEST OR QUESTION. THE RESULT OF THE INPUT IS PLACED IN W OR Z. THE FORM OF THE REQUESTS AND ANSWERS IS BEYOND THE SCOPE OF THIS PAPER.

<STOP STATEMENT>A ::= $STOP$; B ::= $STOP $ <CLEAN VARIABLE>V; C ::= $STOP $ <ALPHA CONSTANT>C

(TYPE[V] = ALPHA)

EFFECT[A(S)] = S; EFFECT[B(S)] = S; EFFECT[C(S)] = S

EXIT[A(S)] = STOP; EXIT[B(S)] = NORMAL; EXIT[C(S)] = NORMAL

THE FIRST FORM TERMINATES THE PROGAM NORMALLY. (NOTE: A STOP IS ADDED AFTER THE FINAL "END" IN A PROGRAM BY THE COMPILER.) THE OTHER FORMS DISPLAY VALUE[V(S)] OR VALUE[L] TO THE SYSTEM OPERATOR AND WAIT FOR A RESPONSE. THE RESPONSE IS NOT PROVIDED TO THE PROGRAM.

<STOQUE STATEMENT> -- BEYOND THE SCOPE OF THIS PAPER. (A STORAGE QUEUE METHOD OF INTER-PROGRAM COMMUNICATION).

<TRACE STATEMENT>A ::= $TRACE $ <BOOLEAN CONSTANT>B

EFFECT[A(S)] = S

EXIT[A(S)] = NORMAL

IF VALUE[B] = TRUE THEN TRACE IS TURNED ON (AN INSTRUCTION BY INSTRUCTION TRACE OF THE PROGRAM IS PRINTED SHOWING THE RESULT OF EACH INSTRUCTION); IF VALUE[B] = FALSE THEN THE TRACE IS TURNED OFF.

<UNLOCK STATEMENT>A ::= $UNLOCK $ <FILE NAME>F

EFFECT[A(S)] = S

EXIT[A(S)] = NORMAL

VALID FOR DISK FILES ONLY. UNLOCKS A RECORD THAT HAD BEEN READ WITH LOCK. THE FILE MUST BE OPEN.

<WRITE STATEMENT>A ::= $WRITE $ <FILE NAME>F <LOCK>;
B ::= $WRITE $ <FILE NAME>G <LOCK> $[$ <LABEL>L $]$;
C ::= $WRITE $ <FILE NAME>H <PRINTER OPTION>
D ::= $WRITE $ <FILE NAME>I <PRINTER OPTION>
$[$ <LABEL>M $]$

EFFECT[A(S)] = S; EFFECT[B(S)] = S; EFFECT[C(S)] = S; EFFECT[D(S)] = S

IF NOT EOF THEN OUTPUT[F] = RECORD[F]; IF NOT EOF THEN OUTPUT[G] = RECORD[G]; IF NOT EOF THEN OUTPUT[H] = RECORD[H]; IF NOT EOF THEN OUTPUT[I] = RECORD[I]

EXIT[A(S)] = IF EOF THEN ABORT ELSE NORMAL;
EXIT[B(S)] = IF EOF THEN LABEL[L] ELSE NORMAL;
EXIT[C(S)] = NORMAL;
EXIT[D(S)] = IF END-OF-PAGE THEN LABEL[M] ELSE NORMAL

FORMS C & D ARE VALID FOR PRINT FILES ONLY.

<PRINTER OPTION> ::= <EMPTY> // $ PAGE$ // $ SINGLE$ //
$ DOUBLE$ // $ NU$ // <INTEGER>I
PAGE CONTROLS: <EMPTY> IS THE SAME AS "DOUBLE"
(SPACING). "NU" MEANS NO SPACING (OVERPRINT).
<INTEGER> IS SKIP-TO-CHANNEL NUMBER (01 < VALUE[I] <
12). NOTE: LINE IS PRINTED, THEN SPACING IS DONE.

<ZIP STATEMENT>A ::= $ZIP $ <CLEAN VARIABLE>V;
B ::= $ZIP $ <ALPHA CONSTANT>C

EFFECT[A(S)] = S; EFFECT[B(S)] = S

EXIT[A(S)] = NORMAL; EXIT[B(S)] = NORMAL

(TYPE[V] = ALPHA)

THE VALUE[V(S)] OR VALUE[C] IS HANDLED AS A MCP CONTROL INSTRUCTION AND PROCESSED ACCORDINGLY.

NOTE THAT THE EXACT EFFECT OF MANY OF THESE STATEMENTS DEPENDS ON S IN NON-TRIVIAL WAYS. INDIVIDUAL CASES CAN BE ANALYSED UNDER FIXED CONDITIONS, BUT GENERAL STATEMENTS MUST BE VAGUE. FOR EXAMPLE, IT IS POSSIBLE TO CHANGE THE STATE VECTOR OF A PROGRAM BY A ZIP STATEMENT (EITHER THE ZIPPING PROGRAM OR SOME OTHER PROGRAM).

<MPC STATEMENT>A ::= <ACCEPT STATEMENT>B;
C ::= <CLOSE STATEMENT>D;
E ::= <DATA COMM I/O STATEMENT>F;
G ::= <DISPLAY STATEMENT>H;
I ::= <DOZE STATEMENT>J;
K ::= <DUMP STATEMENT>L;
M ::= <FILL STATEMENT>N;
O ::= <MICR STATEMENT>P;
Q ::= <OPEN STATEMENT>R;
S ::= <READ STATEMENT>T;
U ::= <SEEK STATEMENT>V;
W ::= <SORT STATEMENT>X;
Y ::= <SPOMESSAGE STATEMENT>Z;

```
AB ::= <STOP STATEMENT>CD;
EF ::= <STOQUE STATEMENT>GH;
IJ ::= <TRACE STATEMENT>KL;
MN ::= <UNLOCK STATEMENT>OP;
QR ::= <WRITE STATEMENT>ST;
UV ::= <ZIP STATEMENT>WX

EFFECT[A(S)] = EFFECT[B(S)]; EFFECT[C(S)] = EFFECT[D(S)];
     EFFECT[E(S)] = EFFECT[F(S)]; EFFECT[G(S)] =
     EFFECT[H(S)]; EFFECT[I(S)] = EFFECT[J(S)];
     EFFECT[K(S)] = EFFECT[L(S)]; EFFECT[M(S)] =
     EFFECT[N(S)]; EFFECT[O(S)] = EFFECT[P(S)];
     EFFECT[Q(S)] = EFFECT[R(S)]; EFFECT[S(S)] =
     EFFECT[T(S)]; EFFECT[U(S)] = EFFECT[V(S)];
     EFFECT[W(S)] = EFFECT[X(S)]; EFFECT[Y(S)] =
     EFFECT[Z(S)]; EFFECT[AB(S)] = EFFECT[CD(S)];
     EFFECT[EF(S)] = EFFECT[GH(S)]; EFFECT[IJ(S)] =
     EFFECT[KL(S)]; EFFECT[MN(S)] = EFFECT[OP(S)];
     EFFECT[QR(S)] = EFFECT[ST(S)]; EFFECT[UV(S)] =
     EFFECT[WX(S)]

EXIT[A(S)] = EXIT[B(S)]; EXIT[C(S)] = EXIT[D(S)];
     EXIT[E(S)] = EXIT[F(S)]; EXIT[G(S)] = EXIT[H(S)];
     EXIT[I(S)] = EXIT[J(S)]; EXIT[K(S)] = EXIT[L(S)];
     EXIT[M(S)] = EXIT[N(S)]; EXIT[O(S)] = EXIT[P(S)];
     EXIT[Q(S)] = EXIT[R(S)]; EXIT[S(S)] = EXIT[T(S)];
     EXIT[U(S)] = EXIT[V(S)]; EXIT[W(S)] = EXIT[X(S)];
     EXIT[Y(S)] = EXIT[Z(S)]; EXIT[AB(S)] = EXIT[CD(S)];
     EXIT[EF(S)] = EXIT[GH(S)]; EXIT[IJ(S)] = EXIT[KL(S)];
     EXIT[MN(S)] = EXIT[OP(S)]; EXIT[QR(S)] = EXIT[ST(S)];
     EXIT[UV(S)] = EXIT[WX(S)]
```

LOOP STATEMENTS

<WHILE STATEMENT>X ::= $WHILE $ <BOOLEAN EXPRESSION>B $ DO $
<STATEMENT>T

THE <WHILE STATEMENT> HAS EXACTLY THE SAME EFFECT AND EXIT INDEX AS THE FOLLOWING SECTION:

```
L:  IF <BOOLEAN EXPRESSION>B THEN
    BEGIN
        <STATEMENT>I;
        GO TO L;
    END
```

NOTE THE <BOOLEAN EXPRESSION> IS COMPLETELY RE-EVALUATED EACH TIME.
WHILE TRUE DO <STATEMENT>
GENERATES AN EFFICIENT INFINITE LOOP WITH NO COMPILER GENERATED CHECKS.

<UNTIL STATEMENT>X ::= $DO $ <STATEMENT>T $ UNTIL $
<BOOLEAN EXPRESSION>B

THE <UNTIL STATEMENT> HAS EXACTLY THE SAME EFFECT AND EXIT INDEX AS THE FOLLOWING SECTION:

```
L:  <STATEMENT>T;
    IF NOT <BOOLEAN EXPRESSION>B THEN GO TO L
```

NOTE THAT THE <BOOLEAN EXPRESSION> IS RE-EVALUATED EACH TIME.

<LOOP STATEMENT>A ::= <WHILE STATEMENT>X;
B ::= <UNTIL STATEMENT>Y

EFFECT[A(S)] = EFFECT[X(S)]; EFFECT[B(S)] = EFFECT[Y(S)]

EXIT[A(S)] = EXIT[X(S)]; EXIT[B(S)] = EXIT[Y(S)]

CMP[A(S)] = CMP[X(S)]; CMP[B(S)] = CMP[Y(S)]

CASE STATEMENT

<CASE STATEMENT>C ::= $CASE $ <VARIABLE>V $ OF $ $BEGIN $
<STATEMENT>S0 $ ELSE $ <STATEMENT>S1 $ ELSE $ <STATEMENT>S2
$ ELSE $ ... <STATEMENT>SN $ END$
D ::= $CASE $ $NO $ <VARIABLE>W $ OF $ $BEGIN $
<STATEMENT>T0 $ ELSE $ <STATEMENT>T1 $ ELSE $ ...
<STATEMENT>TM $ END$

(AS MANY STATEMENTS MAY BE USED IN EITHER FORM AS DESIRED. BY SI OR TI WE MEAN THE I-TH SUCH STATEMENT. THE ... IMPLIES THE PRESENCE OF THE MISSING (IF ANY) STATEMENTS.)

(TYPE[V] = INTEGER OR NUMERIC; TYPE[W] = INTEGER OR NUMERIC)

FOR NOTATIONAL CONVENIENCE, LET
I = MINIMUM[N, VALUE[V(S)]], J = VALUE[W(S)]

EFFECT[C(S)] = EFFECT[SI(SIDE[V(S)])];

EFFECT[D(S)] = IF J ≤ M THEN EFFECT[TJ(SIDE[W(S)])] ELSE
<INDETERMINABLE>

EXIT[C(S)] = EXIT[SI(SIDE[V(S)])];

EXIT[D(S)] = IF J ≤ M THEN EXIT[TJ(SIDE[W(S)])] ELSE
<INDETERMINABLE>

CMP[C(S)] = CMP[SI(SIDE[V(S)])];

CMP[D(S)] = IF J ≤ M THEN CMP[TJ(SIDE[W(S)])] ELSE
<INDETERMINABLE)

(THE FIRST OPTION (C) DOES CHECKING AS TO THE VALUE OF V AND HANDLES ANY OUT OF BOUNDS CONDITIONS [NOTE FOR VALUE[V] > N, STATEMENT SN IS USED]. THE SECOND OPTION (D) DOES NO CHECKING FOR VALUE[W] ≤ M. IF W IS OUT OF BOUNDS, RESULTS DEPEND ON WHAT FOLLOWS THE CASE STATEMENT. MOST OFTEN, THE RESULT IS SOME RUN-TIME PROGRAM ERROR CAUSING THE PROGRAM TO BE SUSPENDED BY THE MCP.)

COMPARE STATEMENT

```
<COMPARE STATEMENT>X ::= $COMPARE $ <BOOLEAN EXPRESSION>B;
     Y ::= $COMPARE $ <ITEM>I $ TO $ <ITEM>J;
     Z ::= $COMPARE $ <LOGICAL TERM>L $ OR $ <LOGICAL TERM>M

     EFFECT[X(S)] = SIDE[B(S)]; EFFECT[Y(S)] =
          SIDE[J(SIDE[I(S)])]; EFFECT[Z(S)] =
          SIDE[M(SIDE[L(S)])]

     EXIT[X(S)] = NORMAL; EXIT[Y(S)] = NORMAL; EXIT[Z(S)] =
          NORMAL

     CMP[X(S)] = CMP[B(S)];
          CMP[Y(S)] = CMP[VALUE[I(S)] = VALUE[J(SIDE[I(S)])]];
          CMP[Z(S)] = CMP[VALUE[L(S)] OR VALUE[M(SIDE[L(S)])]]

          THE COMPARE STATEMENT PRODUCES THE SPECIFIED TEST BUT
          NO CONDITIONAL BRANCH IS PERFORMED.  SEE BOOLEAN
          EXPRESSION AND LOGICAL EXPRESSION ABOVE.
```

CONVERT STATEMENT

<WITH> ::= $ WITH $ // <BLANKS>

<TO> ::= $ TO $ // <BLANKS>

<PICTURE>P ::= <NAME>N

TYPE[P] = ALPHA
(SEE PICTURE DECLARATION)

<EDIT STATEMENT>A ::= $EDIT $ <IDENTIFIER>I <WITH> <PICTURE>P
<TO> <IDENTIFIER>J

EFFECT[A(S)] = S" WHERE S"(J(SIDE[I(S)])) = PICTURE P
APPLIED TO VALUE[I(S)], S"(S) = S OTHERWISE

A <PICTURE> IS A COBOL-LIKE PICTURE. THE DATA IN I IS EDITED ACCORDING TO THE INDICATED PICTURE AND PLACED IN J.
(TYPE[J] = NUMRIC OR ALPHA)

<TRANSLATE STATEMENT>A ::= $TRANSLATE $ <IDENTIFIER>I <WITH>
<IDENTIFIER>T <TO> <IDENTIFIER>J

(TYPE[I] ≠ BOOLEAN; TYPE[J] = ALPHA OR NUMERIC;
TYPE[T] = ALPHA OR NUMERIC; LENGTH[I(S)] =
LENGTH[J(SIDE[T(SIDE[I(S)])])]; ADDRESS[T(SIDE[I(S)])]
= 0 MOD 1000)

EFFECT[A(S)] = S" WHERE S"(J(SIDE[T(SDIE[I(S)])])) =
TRANSLATE TABLE T APPLIED TO VALUE[I(S)], S"(X) = S
OTHERWISE

EXIT[A(S)] = NORMAL

THE TRANSLATE STATEMENT ALLOWS THE TRANSLATION OF A SET OF DATA TO A DIFFERENT SET (IE: FOR CODE CONVERSION). THE EXACT FORM OF THE TRANSLATE TABLE IS BEYOND THE SCOPE OF THIS PAPER.

<CONVERT STATEMENT>A ::= <EDIT STATEMENT>E; B ::=
<TRANSLATE STATEMENT>T

EFFECT[A(S)] = EFFECT[E(S)]; EFFECT[B(S)] = EFFECT[T(S)]

EXIT[A(S)] = NORMAL; EXIT[B(S)] = NORMAL

THE EDIT AND TRANSLATE STATEMENTS ARE DIRECTLY IMPLEMENTED IN THE HARDWARE.

FIND STATEMENT

<NOISE> ::= <BLANKS> // $ FOR $

<EQUAL> ::= <BLANKS> // $ EQUAL $

<FIND TERM>A ::= <CONSTANT>C; B ::= <IDENTIFIER>V

VALUE[A] = VALUE[C]; VALUE[B(S)] = VALUE[V(S)]

TYPE[A] = TYPE[C]; TYPE[B] = TYPE[V]

LENGTH[A] = LENGTH[C]; LENGTH[B(S)] = LENGTH[V(S)]

<SCAN STATEMENT>A ::= $SCAN$ <EQUAL> <FIND TERM>R <NOISE> <FIND TERM>T;
B ::= $SCAN$ $ UNEQUAL $ <FIND TERM>U <NOISE> <FIND TERM>V;
C ::= $SCAN$ $ ZONE$ <EQUAL> <FIND TERM>W <NOISE> <FIND TERM>X;
D ::= $SCAN $ $ZONE$ $ UNEQUAL $ <FIND TERM>Y <NOISE> <FIND TERM>Z

(THE SCAN STATEMENT USES A FIELD IN THE USER=S CORE TO GIVE INFORMATION ON THE RESULT OF THE SCAN. WE SHALL CALL THIS FIELD "SR" (FOR SCAN RESULT). SR HAS THE FOLLOWING ATTRIBUTES: ADDRESS[SR] = 38; TYPE[SR] = INTEGER; LENGTH[SR] = 2 (DIGITS).)

THE SCAN INSTRUCTIONS ARE USED TO SCAN A FIELD FOR A SET OF DELIMETERS (OR THE ABSENCE OF A SET OF DELIMETERS). DEFINE SCAN[A, B] = -1 IF NO CHARACTER/DIGIT OF A = ANY CHARACTER/DIGIT OF B. = 0 IF THE FIRST CHARACTER/DIGIT OF A IS SOME CHARACTER/DIGIT OF B, = 1 IF THE SECOND CHARACTER/DIGIT OF A IS SOME CHARACTER/DIGIT OF B (AND THE FIRST WAS NOT), ..., = N IF THE (N+1)ST CHARACTER/DIGIT OF A IS SOME CHARACTER/DIGIT OF B AND THE FIRST N CHARACTER/DIGITS OF A ARE NOT CHARACTER/DIGITS OF B. (IN THE ABOVE, IF THE TYPE OF A AND B IS ALPHA OR NUMERIC READ CHARACTER FOR CHARACTER/DIGIT ELSE READ DIGIT.)

(TYPE[R] = TYPE[T]; TYPE[U] = TYPE[V]; TYPE[W] = ALPHA OR NUMERIC; TYPE[X] = ALPHA OR NUMERIC; TYPE[Y]

```
     = ALPHA OR NUMERIC;  TYPE[Z] = ALPHA OR NUMERIC)

EXIT[A] = NORMAL;  EXIT[B] = NORMAL;  EXIT[C] = NORMAL;
     EXIT[D] = NORMAL

EFFECT[A(S)] = S" WHERE S"(SR) = IF
     SCAN[VALUE[R(S)], VALUE[T(S)]] = -1 THEN LENGTH[R(S)] -
     1 ELSE SCAN[VALUE[R(S)], VALUE[T(S)]], S"(K) =
     SIDE[T(SIDE[R(S(K))])] OTHERWISE

CMP[A(S)] = IF SCAN[VALUE[R(S)], VALUE[T(S)]] = -1 THEN LESS
     ELSE IF SCAN[VALUE[R(S)], VALUE[T(S)]] = 0 THEN EQUAL
     ELSE GREATER

     DEFINE COMPLEMENT[A] TO BE:  IF TYPE[A] = ALPHA OR
     NUMERIC THEN ALL POSSIBLE EBCDIC CHARACTERS WHICH DO
     NOT APPEAR IN A (THEREFORE LENGTH[COMPLEMENT[A]] = 256
     - LENGTH[A]);  OTHERWISE ALL POSSIBLE DIGITS AND
     UNDIGITS WHICH DO NOT APPEAR IN A (THEREFORE
     LENGTH[COMPLEMENT[A]] = 16 - LENGTH[A]).
     LET N(S) = SCAN[VALUE[U(S)], VALUE[COMPLEMENT[V(S)]]].

EFFECT[B(S)] = S" WHERE S"(SR) = IF N(S) = -1 THEN
     LENGTH[U(S)] - 1 ELSE N(S),  S"(K) =
     SIDE[V(SIDE[U(S(K))])] OTHERWISE

CMP[B(S)] = IF N(S) = -1 THEN LESS ELSE IF N(S) = 0 THEN
     EQUAL ELSE GREATER

     DEFINE ZONE[K] TO BE THE RESULT OF RESETTING ALL THE
     ODD ADDRESS DIGITS OF THE ALPHA OR NUMERIC FIELD K.
     IE:
     ZONE[K] = VALUE[K]  AND  REPEAT[%F0%, LENGTH[K]].

EFFECT[C(S)] = S" WHERE S"(SR) = IF
     SCAN[ZONE[W(S)], ZONE[X(S)]] = -1 THEN LENGTH[W(S)] - 1
     ELSE SCAN[ZONE[W(S)], ZONE[X(S)]], S"(K) =
     SIDE[X(SIDE[W(S(K))])] OTHERWISE

CMP[C(S)] = IF SCAN[ZONE[W(S)], ZONE[X(S)]] = -1 THEN LESS
     ELSE IF SCAN[ZONE[W(S)], ZONE[X(S)]] = 0 THEN EQUAL
     ELSE GREATER

     LET N(S) = SCAN[ZONE[Y(S)], ZONE[COMPLEMENT[Z(S)]]].

EFFECT[D(S)] = S" WHERE S"(SR) = IF N(S) = -1 THEN
```

LENGTH[Y(S)] - 1 ELSE N(S), S"(K) =
SIDE[Z(SIDE[Y(S(K))])] OTHERWISE

CMP[O(S)] = IF N(S) = -1 THEN LESS ELSE IF N(S) = 0 THEN
EQUAL ELSE GREATER

THE SCAN STATEMENT (TYPES A OR B) HAS OBVIOUS USES IN SCANNING DATA FOR CERTAIN CHARACTERS. NOTE THAT IT STOPS WHEN THE FIRST MATCH (OR NON-MATCH) IS FOUND, PROVIDES A POINTER TO THE CHARACTER/DIGIT FOUND, AND THE COMPARISON INDICATOR INDICATES WHETHER THE SCAN WAS SUCCESSFUL. THE SCAN ZONE FORMS (C AND D) ARE USEFUL FOR FINDING CLASSES OF DATA. FOR EXAMPLE, THE CHARACTER ZERO IS A F0 IN EBCDIC. ALL NUMBERS N ARE FN IN EBCDIC. SINCE THE OTHER SIX EBCDIC CHARACTERS WHICH START WITH F ARE NON-PRINTABLE (FA, FB, FC, FD, FE, AND FF), A CHECK FOR A NON-NUMBER IN A FIELD CALLED SOCIALSECURITYNUMBER CAN BE DONE BY
SCAN ZONE UNEQUAL SOCIALSECURITYNUMBER FOR "0";
IF LEQ THEN & HAVE A NON-NUMBER IN THE FIELD.

<SEARCH FIELD>A ::= <IDENTIFIER>I;
B ::= <IDENTIFIER>J $ THRU $ <IDENTIFIER>K

(AT MOST ONE OF J OR K CAN BE AN <ARRAY VARIABLE>. NONE OF I, J, OR K MAY HAVE AN <TYPE MODIFIER> OF "IA". TYPE[I] ≠ BOOLEAN; TYPE[J] ≠ BOOLEAN; TYPE[K] ≠ BOOLEAN)

THE SEARCH STATEMENT WORKS FROM A START ADDRESS TO AN END ADDRESS:

STARTADDRESS[A(S)] = ADDRESS[I(S)]; STARTADDRESS[B(S)] =
ADDRESS[J(S)]

ENDADDRESS[A(S)] = IF (TYPE[I] = ALPHA OR NUMERIC) THEN
ADDRESS[I(S)] + 2×LENGTH[I(S)] ELSE IF TYPE[I] = SIGNED
THEN ADDRESS[I(S)] + LENGTH[I(S)] + 1 ELSE
ADDRESS[I(S)] + LENGTH[I(S)];
ENDADDRESS[B(S)] = ADDRESS[K(SIDE[J(S)])]

TYPE[A] = TYPE[I]; TYPE[B] = TYPE[J]

SIDE[A(S)] = SIDE[I(S)]; SIDE[B(S) = SIDE[K(SIDE[J(S)])]

<INCREMENT>A ::= <EMPTY>; B ::= $ INCREMENT $ <INTEGER>I;
C ::= $ INCREMENT $ $ADDRESS $ <VARIABLE>V;
D ::= $ INCREMENT $ <VARIABLE>W

THE SEARCH STATEMENT NEEDS AN INCREMENT VALUE DEPENDING ON THE TYPE OF THE SEARCH FIELD. LET SFTYPE = IF THE SEARCH FIELD TYPE IS NUMERIC OR ALPHA THEN ALPHA ELSE IF THE SEARCH FIELD TYPE IS SIGNED THEN SIGNED ELSE INTEGER.

INCREMENT[A] = IF SFTYPE = INTEGER THEN 1 ELSE 2;
INCREMENT[B] = IF SFTYPE = ALPHA THEN 2×VALUE[I] ELSE IF SFTYPE = SIGNED THEN VALUE[I] + 1 ELSE VALUE[I];
INCREMENT[C(S)] = IF SFTYPE = ALPHA THEN 2×VALUE[V(S)] ELSE IF SFTYPE = SIGNED THEN VALUE[V(S)] + 1 ELSE VALUE[V(S)];
INCREMENT[D(S)] = ADDRESS[W] - STARTADDRESS[S]

(VALUE[I] < 100; V CAN NOT BE INDEXED; V CAN NOT HAVE A <TYPE MODIFIER> OF "IA"; ADDRESS[V] < 40; ADDRESS[V] = 0 MOD 2; LENGTH[V] = 2 [INDIRECT FIELD LENGTH AGAIN]; W CAN NOT BE INDEXED; W CAN NOT HAVE A <TYPE MODIFIER> OF "IA"; ADDRESS[W] > STARTADDRESS[S])

<SEARCH STATEMENT>A ::= $SEARCH$ <EQUAL> <SEARCH FIELD>T <NOISE> <FIND TERM>U <INCREMENT>I;
B ::= $SEARCH$ $ LOW $ <SEARCH FIELD>V <NOISE> <FIND TERM>W <INCREMENT>J;
C ::= $SEARCH$ $ LOWEST $ <SEARCH FIELD>X <NOISE> <FIND TERM>Y <INCREMENT>K

(TYPE[T] = TYPE[U]; TYPE[V] = TYPE[W]; TYPE[X] = TYPE[Y])

THE SEARCH STATEMENTS HAVE EXACTLY THE SAME EFFECT AS THE FOLLOWING PSEUDO SECTIONS. ALL VARIABLES WHICH START WITH "X" ARE NOT IN THE USER-S SPACE. THE STATEMENT ENDSTATEMENT[COMPARISON] IS USED IN INDICATE THE END OF THE SEARCH STATEMENT PROCESSING AND THE VALUE OF THE COMPARISON INDICATORS AT THE CONCLUSION OF THE STATEMENT.

```
CASE A:
      XA ← STARTADDRESS[T(S)];
      WHILE XA < ENDADDRESS[T(S)] DO
      BEGIN
            IF VALUE[U(SIDE[T(S)])] =
                  CONTENT[XA, LENGTH[U(SIDE[T(S)])], TYPE[U]] THEN
                  BEGIN
                  IX1 ← XA;
                  ENDSTATEMENT[EQUAL];
            END;
            XA ← XA + INCREMENT[I(S)];
      END;
      ENDSTATEMENT[GREATER];


CASE B:
      XA ← STARTADDRESS[V(S)];
      WHILE XA < ENDADDRESS[V(S)] DO
      BEGIN
            IF VALUE[W(SIDE[V(S)])] >
                  CONTENT[XA, LENGTH[W(SIDE[V(S)])], TYPE[W]] THEN
                  BEGIN
                  IX1 ← XA;
                  ENDSTATEMENT[EQUAL];
            END;
            XA ← XA + INCREMENT[J(S)];
      END;
      ENDSTATEMENT[GREATER];


CASE C:
      XF ← 0;
      XL ← ADDRESS[Y(SIDE[X(S)])];
      XA ← STARTADDRESS[X(S)];
      WHILE XA < ENDADDRESS[X(S)] DO
      BEGIN
            IF CONTENT[XL, LENGTH[Y(SIDE[(S)])], TYPE[Y]] >
                  CONTENT[XA, LENGTH[Y(S)])], TYPE[Y]] THEN BEGIN
                  XF ← 1;
                  XL ← XA;
            END;
            XA ← XA + INCREMENT[K(S)];
      END;
      IF XF = 0 THEN ENDSTATEMENT[GREATER];
```

```
IX1 ← XL;
ENDSTATEMENT[EQUAL];
```

IN WORDS, THE SEARCH STATEMENT ALLOWS A FIELD TO BE SEARCHED FOR A SPECIFIC WORD (CASE A), AN ITEM SMALLER THEN THE SPECIFIED WORD (CASE B), OR THE SMALLEST ITEM IN A LIST. IF THE INCREMENT IS 1, THE SEARCH STATEMENT SCANS DOWN THE SEARCH FIELD TRYING EACH SUCCESSIVE GROUP OF CHARACTERS/DIGITS FOR THE SPECIFIED ITEM. IF THE SPECIFIED CONDITION IS SATISFIED (IF AN ITEM IN THE SEARCH FIELD SATISFIES THE CONDITIONS), THE COMPARISON INDICATOR IS SET TO EQUAL AND IX1 POINTS TO THE SATISFYING FIELD.

```
<FIND STATEMENT>A ::= <SCAN STATEMENT>T;
     B ::= <SEARCH STATEMENT>U

     EFFECT[A(S)] = EFFECT[T(S)]; EFFECT[B(S)] = EFFECT[U(S)]

     EXIT[A(S)] = NORMAL; EXIT[B(S)] = NORMAL

     CMP[A(S)] = CMP[T(S)]; CMP[B(S)] = CMP[U(S)]
```

THE SCAN AND SEARCH STATEMENTS ARE DIRECTLY IMPLEMENTED IN THE HARDWARE.

DECLARATIONS

THE DECLARATIONS DEFINE THE ENVIRONMENT OF THE BLOCK (PROGRAM). WE DEFINE ENVIR[] (OF A BLOCK) TO BE THE SET OF ORDERED PAIRS OF ALL <NAME>S DECLARED WITHIN THAT BLOCK, EACH WITH ITS SET OF ATTRIBUTES. TOGETHER WITH THE BLOCK NUMBER. NOTE THE ENVIRONMENT OF A BLOCK B (AS OPPOSED TO ENVIR[B]) IS COMPOSED OF ENVIR[B] .U. ENVIR[X] FOR ALL BLOCKS X WITHIN WHICH B IS NESTED (SEE SCOPE OF IDENTIFIERS).

TO DEFINE THE ATTRIBUTES OF AN IDENTIFIER, WE INTRODUCE
ATTRIBUTE[<IDENTIFIER>I] = (A=ADDRESS[I], L=LENGTH[I],
T=TYPE[I], I=<INITIAL VALUE OF I>, D=DIMENSION[I])
WHICH COMPLETELY DEFINES I. "A=COMPILED" MEANS THE ADDRESS AS DETERMINED BY THE COMPILER (USUALLY THE NEXT AVAILABLE LOCATION WITH NECESSARY SYNC OPERATIONS PEFORMED). "I=DEFAULT" FOR THE INITIAL VALUE MEANS THE UNSPECIFIED COMPILER PRESET VALUE (USUALLY 0 FOR INTEGER OR SIGNED, " " FOR NUMERIC OR ALPHA [NOTE " " HAS A CODE OF %40%, THEREFORE #" " = 0]).

REPEATING THE SYNTAX DEFINITION OF A <BLOCK>:

<BLOCK>X ::= $BEGIN $ <DECLARATIONS>D <SECTION>T $ END$

ENVIR[X] = ENVIR[D]

<DECLARATIONS>A ::= <DECLARATION>D; B ::= <DECLARATIONS>E
<DECLARATION>F

ENVIR[A] = ENVIR[D]; ENVIR[B] = ENVIR[E] .U. ENVIR[F]

<DECLARATION>A ::= <SIMPLE DECLARATION>E;
B ::= <ARRAY DECLARATION>R;
C ::= <LABEL DECLARATION>L;
D ::= <PROCEDURE DECLARATION>P;
X ::= <FILE DECLARATION>F

ENVIR[A] = ENVIR[E]; ENVIR[B] = ENVIR[R]; ENVIR[C] = ENVIR[L]; ENVIR[D] = ENVIR[P]; ENVIR[X] = ENVIR[F]

THE FOLLOWING SECTIONS DISCUSS THESE TYPES OF DECLARATIONS INDIVIDUALLY (FILE DECLARATIONS ARE BEYOND THE SCOPE OF THIS

PAPER).

SIMPLE DECLARATIONS

```
<BIT ITEM>A ::= <NAME>N;
     B ::= <NAME>M $←$ <BOOLEAN CONSTANT>C

     ATTRIBUTE[A] = (A=COMPILED, L=1 BIT, T=BOOLEAN, I=FALSE,
          D=1);

     ATTRIBUTE[B] = (A=COMPILED, L=1 BIT, T=BOOLEAN, I=VALUE[C],
          D=1)

     ENVIR[A] = (N, ATTRIBUTE[A]);  ENVIR[B] = (M, ATTRIBUTE[B])


<BIT LIST>A ::= <BIT ITEM>I;
     B ::= <BIT LIST>L $,$ <BIT ITEM>J

     ENVIR[A] = ENVIR[I];  ENVIR[B] = ENVIR[L] .U. ENVIR[J]


<BIT DECLARATION>A ::= $BIT $ <BIT LIST>L $;$

     ENVIR[A] = ENVIR[L]


<PICTURE DECLARATION>A ::= $PICTURE $ <NAME>N $←$
     $"$ <COBOL PICTURE>P $"$ $;$

     ATTRIBUTE[A] = (A=COMPILED, L=LENGTH[MICRO-OPS[P]], T=ALPHA,
          I=MICRO-OPS[P], D=1)

     ENVIR[A] = (N, ATTRIBUTE[A])

          THE EXACT FORMS OF A <COBOL PICTURE> AND THE MICRO-OPS
          GENERATED ARE BEYOND THE SCOPE OF THIS PAPER.  THE USE
          OF A <PICTURE DECLARATION> OR AN <EDIT STATEMENT>
          REQUIRES THAT
               CONTENT[48, ALPHA, 8] = "+-*.,$0 " (SUPPLIED BY
          THE COMPILER).


<SYNC>A ::= <EMPTY>;  B ::= $ MOD $ <INTEGER>I

          (0 < VALUE[I] < 1,000,000)
```

SYNC[A] = 1; SYNC[B] = VALUE[I]

DEFINE MOD[X, I] = Y SUCH THAT Y = 0 MOD I, X ≤ Y, AND IF Z = 0 MOD I AND Z ≤ Y THEN Y ≤ Z (IE: THE NEXT NUMBER STARTING WITH X THAT IS 0 MOD I).

<DATA RIGHT PART>A ::= <SYNC>Y;
B ::= $=$ <CLEAN VARIABLE>V;
C ::= <SYNC>Z $←$ <CONSTANT>C

TYPE[] (TYPE OF DECLARATION TO WHICH THIS BELONGS)

LENGTH[] (LENGTH OF THE DECLARATION TO WHICH THIS BELONGS).

ADDRESS[A] = MOD[COMPILED, SYNC[Y]]; ADDRESS[B] = ADDRESS[V]; ADDRESS[C] = MOD[COMPILED, SYNC[Z]]

INITIAL[A] = DEFAULT; INITIAL[B] = DEFAULT; INITIAL[C] = VALUE[C]

(TYPE[C] = TYPE[]; LENGTH[C] ≤ LENGTH[])

<DATA LEFT PART>A ::= <NAME>N $($ <INTEGER>I $)$

LENGTH[A] = VALUE[I]

NAME[A] = N

(0 < VALUE[I] < 1,000,000)

<DATA ELEMENT>A ::= <DATA LEFT PART>L <DATA RIGHT PART>R

TYPE[] (TYPE OF DELCARATION TO WHICH THIS BELONGS)

ATTRIBUTE[A] = (A=ADDRESS[R], L=LENGTH[L], T=TYPE[], I=INITIAL[R], D=1)

ENVIR[A] = (NAME[L], ATTRIBUTE[A])

<DATA ELEMENT LIST>A ::= <DATA ELEMENT>D;
B ::= <DATA ELEMENT LIST>L $,$ <DATA ELEMENT>E

```
     ENVIR[A] = ENVIR[D]; ENVIR[B] = ENVIR[L] .U. ENVIR[E]

<VARIABLE DATA TYPE>I ::= $INTEGER $; S ::= $SIGNED INTEGER $;
     A ::= $ALPHA $; N ::= $NUMERIC $; R ::= $REAL $; D ::=
     $DOUBLE $

     TYPE[I] = INTEGER; TYPE[S] = SIGNED; TYPE[A] = ALPHA;
          TYPE[N] = NUMERIC; TYPE[R] = REAL; TYPE[D] = SIGNED

<VARIABLE DECLARATION>A ::= <VARIABLE DATA TYPE>T
     <DATA ELEMENT LIST>L $;$

     ENVIR[A] = ENVIR[L]

<FIXED DATA TYPE>I ::= $FIXED INTEGER $; D ::= $FIXED DOUBLE $;
     R ::= $FIXED REAL $; A ::= $INDIRECT $

     TYPE[I] = SIGNED; TYPE[D] = SIGNED; TYPE[R] = REAL;
          TYPE[A] = INTEGER

     LENGTH[I] = 7; LENGTH[D] = 19; LENGTH[R] = 11; LENGTH[A]
          = 6 (8 IF ON A B4700)

<FIXED DATA ELEMENT>A ::= <NAME>N <DATA LEFT PART>L

     TYPE[]

     LENGTH[]

     ATTRIBUTE[A] = (A=IF TYPE[] = INTEGER THEN MOD[ADDRESS[L],2]
          ELSE MOD[ADDRESS[L],4], L=LENGTH[], T=TYPE[],
          I=INITIAL[L], D=1)

     ENVIR[A] = (N, ATTRIBUTE[A])

<FIXED DATA ELEMENT LIST>A ::= <FIXED DATA ELEMENT>D;
     B ::= <FIXED DATA ELEMENT LIST>L $,$ <FIXED DATA ELEMENT>E

     ENVIR[A] = ENVIR[D]; ENVIR[B] = ENVIR[L] .U. ENVIR[E]
```

```
<FIXED DECLARATION>A ::= <FIXED DATA TYPE>T
     <FIXED DATA ELEMENT LIST>L $;$

     ENVIR[A] = ENVIR[L]


<SIMPLE DECLARATION>A ::= <BIT DECLARATION>T;
     B ::= <PICTURE DECLARATION>P;
     C ::= <VARIABLE DECLARATION>V;
     D ::= <FIXED DECLARATION>F

     ENVIR[A] = ENVIR[T]; ENVIR[B] = ENVIR[P]; ENVIR[C] =
          ENVIR[V]; ENVIR[D] = ENVIR[F]
```

ARRAY DECLARATIONS

```
<ARRAY PRESET LIST>A ::= <CONSTANT>C;
     B ::= <ARRAY PRESET LIST>L $,$ <CONSTANT>D

     TYPE[]

     LENGTH[]  (THE LENGTH OF ONE ARRAY ELEMENT)

     DIMENSION[A] = 1;  DIMENSION[A] = 1;  DIMENSION[B] =
          DIMENSION[L] + 1

     INITIAL[A] = VALUE[C];
          INITIAL[B] = INITIAL[L] .U. VALUE[D]

          (TYPE[C] = TYPE[];  TYPE[D] = TYPE[];  LENGTH[C] ≤
          LENGTH[];  LENGTH[D] ≤ LENGTH[])

          IN THIS CASE, INITIAL IS A SET OF VALUES CORRESPONDING
          OT THE 0TH, 1ST, 2ND, ... ETC ELEMENTS OF THE ARRAY.


<ARRAY RIGHT PART>A ::= <SYNC>Y;
     B ::= $=$ <CLEAN VARIABLE>V;
     C ::= <SYNC>Z $←$ <ARRAY PRESET LIST>L

     TYPE[]

     LENGTH[]

     ADDRESS[A] = MOD[COMPILED, SYNC[Y]];  ADDRESS[B] =
          ADDRESS[V];  ADDRESS[C] = MOD[COMPILED, SYNC[Z]]

     INITIAL[A] = DEFAULT;  INITIAL[B] = DEFAULT;  INITIAL[C] =
          INITIAL[L]

     DIMENSION[A] = NONE;  DIMENSION[B] = NONE;  DIMENSION[C] =
          DIMENSION[L]


<APRAY DECLARATION>A ::= <VARIABLE DATA TYPE>T $ARRAY $ <NAME>N
     $[$ <INTEGER>I $]$ $($ <INTEGER>J $)$ <ARRAY RIGHT PART>R

          (0 < VALUE[I] < 1,000,000;  0 < VALUE[J] < 1,000,000;
          VALUE[R] = TYPE[T];  IF DIMENSION[R] ≠ NONE THEN
```

```
DIMENSION[R] = VALUE[I] + 1)

ATTRIBUTE[A] = (A=ADDRESS[R], L=VALUE[J], T=TYPE[T],
    I=INITIAL[R], D=VALUE[I] + 1)

    (I INDICATES THE HIGHEST VALID SUBSCRIPT VALUE)

ENVIR[A] = (N, ATTRIBUTE[A])
```

LABEL DECLARATION

<LABEL ELEMENT>A ::= <LABEL>L

ATTRIBUTE[A] = (A=COMPILED, L=0, T=INTEGER,, D=1)

ENVIR[A] = (L, ATTRIBUTE[A])

<LABEL LIST>A ::= <LABEL ELEMENT>E; B ::= <LABEL LIST>L $,$ <LABEL ELEMENT>F

ENVIR[A] = ENVIR[E]; ENVIR[B] = ENVIR[L] .U. ENVIR[F]

<LABEL DECLARATIUN>A ::= $LABEL $ <LABEL LIST>L $;$

ENVIR[A] = ENVIR[L]

PROCEDURE DECLARATION

<PROCEDURE BODY>X ::= <BLOCK>B; Y ::= <COMPOUND STATEMENT>C

EFFECT[X(S)] = EFFECT[B(S)]; EFFECT[Y(S)] = EFFECT[C(S)]

EXIT[X(S)] = EXIT[B(S)]; EXIT[Y(S)] = EXIT[C(S)]

EACH CALL ON A PROCEDURE ADDS ANOTHER LEVEL TO THE PROGRAM-S STACK; EACH RETURN FROM A PROCEDURE REMOVES THE TOP ENTRY FROM THE PROGRAM STACK. EACH STACK ENTRY CONTAINS ALL PASSED PARAMETERS AND INFORMATION ABOUT WHERE TO RETURN TO. THEREFORE, PROCEDURES CAN BE CALLED RECURSIVELY OR NESTED WITH NO RETURN PROBLEMS. DECLARATIONS WITHIN A PROCEDURE BODY AS DEFINED IN DECLARATIONS ABOVE CAN NOT BE PRESET SINCE THEY ALSO APPEAR IN THE STACK (TO ALLOW RECURSIVENESS). IF DECLARED ITEMS ARE NOT TO BE CHANGED (OR THE PROCEDURE IS NOT CALLED RECURSIVELY), "OWN " MAY PRECEED THE DECLARATION. IN THIS CASE THE DATA MAY BE PRESET SINCE THE ITEM IS NOT PLACED IN THE STACK. ALL FILE AND FILE RECORD DECLARATIONS IN A PROCEDURE BODY MUST BE "OWN". THE INITIAL VALUE OF NON-OWN ITEMS IS <INDETERMINABLE>.

THE HARDWARE USES IX3 TO POINT TO THE CURRENT STACK LEVEL AND BASE+40 FOR 6 DIGITS TO POINT TO THE NEXT STACK LEVEL. THE CHANGING OF EITHER OF THESE FIELDS WITHIN A PROCEDURE WILL PRODUCE INDETERMINABLE RESULTS (USUALLY). THE INITIAL VALUE OF BASE+40 IS SET UP BY THE COMPILER.

A SPECIAL STATEMENT CAN BE USED TO RETURN FROM A PROCEDURE (THE FINAL "END" OF A PROCEDURE ACTS AS A RETURN):

<EXIT STATEMENT>A ::= $EXIT$; B ::= $EXIT$ <TO> <LABEL>L; C ::= $EXIT$ <TO> <VARIABLE>V

ALL THREE FORMS POP THE STACK. THE FIRST FORM IS A NORMAL SUBROUTINE RETURN. THE LAST TWO ACT LIKE A <TRANSFER STATMENT> OF THE SAME FORM IN ADDITION TO POPPING THE STACK. THE USE OF AN <EXIT STATEMENT> OUTSIDE OF A PROCEDURE WILL HAVE INDETERMINABLE RESULTS (USUALLY).

<SEGMENTED> ::= <EMPTY> // $SEGMENTED $

<PERIOD> ::= <BLANKS> // $.$

<PARAMETER LIST>A ::= <NAME>N; B ::= <PARAMETER LIST>L $,$ <NAME>M

NUMARGS[A] = 1; NUMARGS[B] = NUMARGS[L] + 1
(NUMARGS[L] ≤ 9)

ARGNUM[N] = 1; ARGNUM[M] = NUMARGS[L] + 1

<VALUE LIST> ::= <EMPTY> // $VALUE $ <PARAMETER LIST> $;$

<FORMAL PARAMETER PART>A ::= <EMPTY>; B ::= $($ <PARAMETER LIST>L $)$ <VALUE LIST>V <DECLARATIONS>D

NUMARGS[A] = 0; NUMARGS[B] = NUMARGS[L]

EVERY PARAMETER IN L MUST BE DECLARED IN D (AND NOTHING ELSE MAY BE DECLARED IN D). PARAMETERS CAN BE EITHER CALL-BY-NAME OR CALL-BY-VALUE. EACH CALL-BY-VALUE PARAMETER MUST APPEAR IN V. THE VALUE OF THE CALLING ARGUMENT IS PASSED FOR A CALL-BY-VALUE PARAMETER; THE ADDRESS OF THE CALLING PARAMETER IS PASSED FOR A CALL-BY-NAME PARAMETER. THE COMPILER HANDLES ALL INDIRECTION NECESSARY DUE TO CALL-BY-NAME PARAMETERS. IF A CALL-BY-NAME PARAMETER IS ON THE LEFT SIDE OF THE $←$ IN AN ASSIGNMENT STATEMENT, THE VALUE TO THE CALLER IS CHANGED. FOR A CALL-BY-VALUE PARAMETER, IT CAN NOT BE CHANGED AS THE CALLER SEES IT BUT CAN BE CHANGED AS THE PROCEDURE SEES IT.

<PROCEDURE DECLARATION>A ::= $FORWARD PROCEDURE $ <NAME>N <FORMAL PARAMETER PART>F; B ::= <SEGMENTED> $PROCEDURE$ <PERIOD> <NAME>M <FORMAL PARAMETER PART>G <PROCEDURE BODY>P

THE FORWARD PROCEDURE DECLARATION IS USED TO SPECIFY THE EXISTANCE OF A PROCEDURE WHOSE PROCEDURE BODY CAN NOT APPEAR YET (AS WHEN PROCEDURE A CALLS PROCEDURE B AND PROCEDURE B CALLS PROCEDURE A; ONE OF A OR B MUST

BE DELCARED FORWARD). IF <SEGMENTED> IS "SEGMENTED" THIS PROCEDURE FORMS A SEPERATE SEGMENT (SEGMENTATION IS BEYOND THE SCOPE OF THIS PAPER). IF <PERIOD> IS "." THEN THIS PROCEDURE MAY NOT CALL OTHER PROCEDURES: THE COMPILER WILL NOT INCREASE THE STACK ENTRY SIZE TO INCLUDE THE NON-OWN LOCAL VARIABLES.

SCOPE OF IDENTIFIERS

RULE 1: EVERY IDENTIFIER MUST BE DELCARED BEFORE IT CAN BE REFERENCED (EXCEPT WITHIN A <DEFINE STRING>).

EACH BLOCK AUTOMATICALLY INTRODUCES A NEW LEVEL OF NOMENCLATURE. THEREFORE, ANY NAMED DECLARATION OCCURRRING WITHIN THF BLOCK IS SAID TO BE LOCAL TO THE BLOCK IN QUESTION. SUCH A DELCARATION MEANS --

A. THE ENTITIY REPRESENTED BY THE IDENTIFIER INSIDE THE BLOCK HAS NO EXISTANCE OUTSIDE THE BLOCK.
B. ANY ENTITY REPRESENTED BY THE SAME IDENTIFIER OUTSIDE THE BLOCK IS COMPLETELY INACCESSIBLE INSIDE THE BLOCK.

AN IDENTIFIER OCCURRING WITHIN AN INNER BLOCK AND NOT DELCARED WITHIN THAT BLOCK WILL BE NONLOCAL (OR GLOBAL) TO IT; THAT IS, THE IDENTIFIER WILL REPRESENT THE SAME ENTITY INSIDE THE BLOCK AND IN THE LEVEL OR LEVELS IMMEDIATELY OUTSIDE IT, UP TO AND INCLUDING THE LEVEL IN WHICH THE IDENTIFIER WAS DECLARED.

SINCE A STATEMENT WITHIN A BLOCK MAY ITSELF BE A BLOCK, THE CONCEPTS OF LOCAL AND NON-LOCAL TO A BLOCK MUST BE UNDERSTOOD RECURSIVELY. THUS AN IDENTIFIER WHICH IS NONLOCAL TO BLOCK A MAY OR MAY NOT BE NONLOCAL TO BLOCK B IN WHICH BLOCK A IS ONE STATEMENT.

BLOCK B IS SAID TO BE NESTED IN BLOCK A IF BLOCK B IS A WHOLLY CONTAINED STATEMENT WITHIN BLOCK A.

BLOCK A AND BLOCK B ARE SAID TO BE DISJOINT IF NEITHER IS A STATEMENT WITHIN THE OTHER.

A LABEL MUST BE DECLARED IN THE HEAD OF THE INNERMOST BLOCK IN WHICH THE ASSOCIATED LABELLED STATEMENT APPEARS. IF ANY STATEMENT IN A PROCEDURE BODY IS LABELLED, THE DECLARATION OF THIS LABEL MUST APPEAR WITHIN THE PROCEDURE BODY.

IDENTIFIERS IN A PROCEDURE BODY WHICH ARE NOT FORMAL PARAMETERS ARE LOCAL TO THAT PROCEDURE AND ARE INACCESSIBLE FROM OUTSIDE THE PROCEDURE.

APPENDIX A: RESERVED WORDS

RESERVED WORDS ORDERED BY LENGTH:

LENGTH 2:

DO GO IA IF IN IO NM NO OF OI OR SN TO UA UN

LENGTH 3:

ALL ANU BIT CHR DIV END IX1 IX2 IX3 JSL JSR MOD
MUL MFC MVD NOT OUT OWN REM TGL TWX WDS XCH ZIP

LENGTH 4:

BASE B500 CASE COPY DATE DISC DISK DOZE DUMP EDIT
ELSE EXIT FILE FILL FLOW LOCK OPEN PAGE POPQ PUTQ
READ REAL REEL SCAN SEEK SORT STOP TAPE THEN TIME
TRUE TT28 WITH

LENGTH 5:

ALPHA ARRAY BEGIN B2500 B3500 B4700 B9350 B9352
CDATE CLOSE ENTER ENTRY FALSE FIXED JDATE LABEL
POLLQ PULLQ PUNCH PURGE PUSHQ SPACE STORE TAPE7
TAPE9 TC500 TC700 TRACE UNTIL VALUE WHILE WRITE

LENGTH 6:

ACCEPT ACCESS ACTION ATTBA1 COMMON CRUNCH DACCUM
DEFINE DEMAND DOUBLE IACCUM LISTER MEMORY RACCUM
READER REMOVE REWIND SEARCH SEGMNO SIGNED SINGLE
SORTER TAPEPE TIME60

LENGTH 7:

ADDRESS CHANNEL COMMENT COMPARE CONTROL DCT2000
DISPLAY FORWARD IBM1030 IBM1050 INTEGER LINKAGE
NUMERIC PICTURE PRINTER PTPUNCH RELEASE REVERSE
SEGDICT SEGMENT

LENGTH 8:

DATACOMM DISKPACK INDIRECT OVERFLOW PTREADER

LENGTH 9:

OLBANKING PROCEDURE REMAINDER SEGMENTED TOUCHTONE
TRANSLATE

LENGTH 10:

DISCONNECT FRIDEN7311 SPOMESSAGE SUBROUTINE

```
LENGTH 11:
     DISPLAYUNIT  EXITROUTINE  ROUTINETYPE  UNSEGMENTED
```

APPENDIX B: EQUIVALENCES

THE FOLLOWING TABLE LISTS THE STRING USED IN THE SYNTAX/SEMANTICS DEFINITION OF BPL AND THE EQUIVALENT STRING(S) THAT CAN BE USED IN THEIR PLACE AT ANY POINT.

SYMBOL AS USED	EQUIVALENT SYMBOL(S)
$+$	<FILLER> $+$ <FILLER> <FILLER> $:=$ <FILLER>
$;$	<FILLER> $;$ <FILLER>
$:$	<FILLER> $:$ <FILLER>
$,$	<FILLER> $,$ <FILLER>
$[$	<FILLER> $[$ <FILLER>
$]$	<FILLER> $]$ <FILLER>
$($	<FILLER> $($ <FILLER>
$)$	<FILLER> $)$ <FILLER>
$+$	<FILLER> $+$ <FILLER>
$-$	<FILLER> $-$ <FILLER>
$*$	<FILLER> $*$ <FILLER> <BLANKS> MUL <BLANKS>
$/$	<FILLER> $/$ <FILLER> <BLANKS> DIV <BLANKS>
$=$	<FILLER> $=$ <FILLER> <BLANKS> EQL <BLANKS>
$ NEQ $	<FILLER> $≤=$ <FILLER> <BLANKS> NEQ <BLANKS>
$<$	<FILLER> $<$ <FILLER> <BLANKS> LSS <BLANKS>

```
$ LEQ $                 <FILLER> $<=$ <FILLER>
                        <BLANKS> $LEQ$ <BLANKS>

$>$                     <FILLER> $>$ <FILLER>
                        <BLANKS> $GTR$ <BLANKS>

$ GEQ $                 <FILLER> $>=$ <FILLER>
                        <BLANKS> $GEQ$ <BLANKS>

$ NOT $                 <FILLER> $≤$ <FILLER>
                        <BLANKS> $NOT$ <BLANKS>

$ AND $                 <BLANKS> $AND$ <BLANKS>

$ OR $                  <BLANKS> $OR$ <BLANKS>

$OVERFLOW$              <BLANKS> $OVERFLOW$ <BLANKS>

$ END$                  <BLANKS> $END$ <FILLER>
                        <FILLER> $;$ <FILLER> $END$
                              <FILLER>
```

APPENDIX C: DEFINES

THE BPL COMPILER HAS IMPLEMENTED A MEANS OF CODING SHORTHAND CALLED "DEFINES". USE OF DEFINES CAN TRANSFORM THE LANGUAGE TO A COMPLETELY DIFFERENT TYPE OF LANGUAGE. DEFINES CAN BE USED TO INCREASE (OR REDUCE) READIBILITY OF A PROGRAM.

<DEFINE PARAMETER LIST> ::= <NAME> // <DEFINE PARAMETER LIST> $,$ <NAME>

<DEFINE ID> ::= <NAME>

<DFFINE LEFT PART> ::= <DEFINE ID> // <DEFINE ID> $($ <DEFINE PARAMETER LIST> $)$

<DEFINE STRING> ::= <ANY STRING OF EBCDIC CHARACTERS NOT CONTAINING A @9F@ OR $#$>

<DEFINE RIGHT PART> ::= $=$ <DEFINE STRING> $#$

<DEFINE LIST> ::= <DEFINE LEFT PART> <DEFINE RIGHT PART> // <DEFINE LIST> $,$ <DEFINE LEFT PART> <DEFINE RIGHT PART>

<DEFINE DECLARATION> ::= $DEFINE $ <DEFINE LIST> $;$

THE DEFINE DECLARATION IS NOT A REAL DECLARATION: IT DOES NOT DETERMINE IF A COMPOUND STATEMENT BECOMES A BLOCK. NO SYNTAX CHECKING IS DONE AT DECLARATION TIME, ONLY WHEN THE DEFINE IS CALLED.

<DEFINE CALL PARAMETER> ::= <ANY STRING NOT CONTAINING $,$ OR @9F@> // $($ <ANY STRING NOT CONTAINING $)$ OR @9F@> $)$

<DEFINE CALL PARAMETER LIST> ::= <DEFINE CALL PARAMETER> // <DEFINE CALL PARAMETER LIST> $,$ <DEFINE CALL PARAMETER>

<DEFINE CALL> ::= <DEFINE ID> //
<DEFINE ID> $($ <DEFINE CALL PARAMETER LIST> $)$

WHEN A <DEFINE ID> IS FOUND IN THE SOURCE STRING, ITS CORRESPONDING <DEFINE STRING> IS CONSIDERED TO APPEAR IN PLACE OF THE <DEFINE CALL>. IF ANY <DEFINE CALL PARAMETERS>S ARE PROVIDED, THE <DEFINE CALL PARAMETER>S REPLACE THE CORRESPONDING DEFINE PARAMETERS WHENEVER THE DEFINE PARAMETERS APPEAR IN THE <DEFINE STRING>. NOTE THAT THE NUMBER OF PARAMETERS DOES NOT NEED TO MATCH. EXTRA <DEFINE CALL PARAMETER>S ARE IGNORED. EXTRA DEFINE PARAMETERS REMAIN UNCHANGED IN THE <DEFINE STRING>.

SYNTAX CHECKING RESUMES AFTER THE DEFINE HAS BEEN EVALUATED AT THE START OF THE NEW STRING.

DEFINES CAN BE NESTED, PARAMETERS CAN BE DEFINES, ETC. CARE SHOULD BE TAKEN TO AVOID DEFINE LOOPS.

NOTE THAT NO SYNTAX CHECKING IS DONE AT DECLARATION TIME. THEREFORE A <DEFINE STRING> MAY CONTAIN AN IDENTIFIER WHICH HAS NOT BEEN DECLARED YET, BUT THAT IDENTIFIER MUST HAVE BEEN DECLARED BEFORE THE DEFINE CALL ON THAT DEFINE IS MADE.

<DEFINE CALL> CAN APPEAR ONLY WHERE A <NAME> WOULD APPEAR IN THE BPL SYNTAX. CONSIDER THE FOLLOWING EXAMPLE:

```
DEFINE PLUS = + #
     SETX(Y) = X + Y #;

X + B PLUS 1;
SETX(B + 1);
```

ASSUMING EVERYTHING IS DECLARED, BOTH STATEMENTS WOULD APPEAR TO HAVE THE SAME EFFECT (X + B + 1). THE FIRST FORM WILL GENERATE A SYNTAX ERROR, THE SECOND WILL NOT.

CARE MUST ALSO BE TAKEN WITH ";" IN DEFINE STRINGS. CONSIDER:

```
DEFINE A = I + 1 + 1 #,
       B = I + I + 1; # ;

IF X THEN A ELSE ...     & OK
IF X THEN B ELSE ...     & ERROR.
```

APPENDIX D

NOTE ABOUT THIS REPORT

THIS DOCUMENT WAS PREPARED BY A PROGRAM DEVELOPED BY THE AUTHOR WHILE HE WAS EMPLOYED BY BURROUGHS CORPORATION ON THE NAVY STOCK POINTS PROGRAM. THE PROGRAM, WRITTEN IN BPL, ALLOWS AUTOMATIC CREATION OF TABS, SPECIAL CONSTRUCTS, A TABLE OF CONTENTS, AND AN INDEX. THE PROGRAM IS CURRENTLY BEING USED TO MAINTAIN IN HOUSE DOCUMENTATION ON VARIOUS DEVELOPING PROJECTS.

REFERENCES

1. BURROUGHS CORPORATION. MEDIUM SYSTEMS BURROUGHS PROGRAMMING LANGUAGE PRELIMINARY INFORMATION MANUAL. 1972.

2. MAURER, W.D. INTRODUCTION TO PROGRAMMING SCIENCE - PART 1: SYNTAX AND SEMANTICS OF PROGRAMMING LANGUAGES. ELECTRONICS RESEARCH LABORATORY, UNIVERSITY OF CALIFORNIA, BERKELEY.

C H A P T E R S I X

A D E S C R I P T I O N O F F O R T R A N

(for the CDC 6400, 6600, etc., using the RUN compiler)

B Y

L A W R E N C E H . Y E U N G

CONSTANTS & EXPRESSIONS

(SYNTAX & SEMANTICS)

1. Fortran Character Set

1.1) <Integer letter> ::= 'I' | 'J' | 'K' | 'L' | 'M' | 'N'

1.2) <real letter> ::= 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'

1.3) <octal digit> a ::= '0' ; b ::= '1' ; c ::= '2' ; d ::= '3' ; e ::= '4' ; f ::= '5' ; g ::= '6' ; h ::= '7'

<value> $a^v = 0$; $b^v = 1$; $c^v = 2$; $d^v = 3$; $e^v = 4$; $f^v = 5$; $g^v = 6$; $h^v = 7$

1.4) <decimal digit> a ::= <octal digit> x ; b ::= '8' ; c ::= '9'

<value> $a^v = x^v$; $b^v = 8$; $c^v = 9$

1.5) <special character> ::= '=' | '+' | '-' | '*' | '/' | '(' | ')' | ',' | '.' | '$' | ' '

1.6) <numeric character> ::= <octal digit> | <decimal digit>

1.7) <alphabetic character> x ::= <integer letter> i ; y ::= <real letter> r

<type> x^t = integer ; y^t = real

1.8) <alphanumeric character> ::= <alphabetic character> | <numeric character>

1.9) <character> ::= <alphanumeric character> | <special character>

1.10) <statement identifier> S ::= <unsigned integer> i .

<length> $s^n = i^n$

$(1 \le i^n \le 5)$

1.11)

1.11) <alphanumeric identifier> x ::= <alphabetic character> a ;
y ::= <alphanumeric identifier> b <alphanumeric character> c

<type> $x^t = a^t$; $y^t = b^t$

<length> $(1 \le x^n \le 7)$

1.12) <identifier> ::= <alphanumeric identifier> | <statement identifier>

2. VARIABLES (Implicit declaration only)

2.1) <integer variable> x ::= <integer letter> ; y ::= <integer variable>
<alphanumeric character>

<type> $x^t = y^t =$ integer

<length> $(1 \le x^n \le 7)$

2.2) <real variable> x ::= <real letter> ; y ::= <real variable>
<alphanumeric character>

<type> $x^t = y^t =$ real

<length> $(1 \le x^n \le 7)$

2.3) <simple variable> x ::= <integer variable> ; y ::= <real variable>

<type> $x^t =$ integer ; $y^t =$ real

2.4) <subscript list> x ::= <unsigned integer> a ; y ::= <unsigned integer> b ',' <unsigned integer> c ; z ::= <unsigned integer> d ',' <unsigned integer> e ',' <unsigned integer> f

<number of elements> $x^s = a^v$; $y^s = b^v \cdot c^v$; $z^s = d^v \cdot e^v \cdot f^v$

$(x^s, y^s, z^s \le 2^{48} - 1)$

<value> $x^v(s) = a^v(s)$; $y^v(s) =$ concat $(b^v(s), ',', c^v(s))$;

2.5) <subscripted variable> x ::= <simple variable> s '(' <subscript list> ')'

<type> $x^t = s^t$

<L-value> $x^l(s) =$ concat $(a, '(', s^v(s), ')')$

2.6) <variable> v ::= <simple variable> s ; w ::= <subscripted variable> t

<type> $v^t = s^t$; $w^t = t^t$

<L-value> $v^l = s^l$; $w^l = u^l$

<value> $v^v(S) = S(v^l(S))$; $w^v(S) = S(w^l(S))$

3. CONSTANTS

3.1) <unsigned octal integer> x ::= 'ϕ' <octal digit> a ; y ::= <unsigned octal integer> b <octal digit> c ; w ::= <octal digit> d 'B' ; z ::= <octal digit> e <unsigned octal digit> f

<length> $x^n = 1$; $y^n = b^n + 1$; $w^n = 1$; $z^n = e^n + 1$

$(6 \le x^n,\ y^n \le 20,\ 1 \le z^n \le 20,\ 1 \le w^n \le 20)$

<value> $x^v = a^v$; $y^v = 8b^v + c^v$; $w^v = d^v$; $z^v = e^v \exp(8, f^n) + f^v$

3.2) <unsigned decimal integer> x ::= <decimal digit> a ; y ::= <unsigned decimal integer> b <decimal digit> c

<length> $x^n = 1$; $y^n = b^n + 1$

$(x^n \le 18)$

<value> $x^v = a^v$; $y^v = 10b^v + c^v$

3.3) <octal integer> x ::= <unsigned octal integer> a ; y ::= '+' <unsigned octal integer> b ; w ::= '−' <unsigned octal integer> c

<length> $x^n = a^n$; $y^n = b^n$; $w^n = c^n$

<value> $x^v = a^v$; $y^v = b^v$; $w^v = -c^v$

3.4) <decimal integer> x ::= <unsigned decimal integer> a ; y ::= '+' <unsigned decimal integer> b ; z ::= '−' <unsigned decimal integer> c

<length> $x^n = a^n$; $y^n = b^n$; $z^n = c^n$

<value> $x^v = a^v$; $y^v = b^v$; $z^v = -c^v$

3.5) <real no. without exponent> x ::= <unsigned decimal integer> a '.' ;
y ::= '.' <unsigned decimal integer> b ; w ::= <unsigned decimal integer> c '.' <unsigned decimal integer> d

<length> $x^n = a^n$; $y^N = b^N$; $w^n = c^n + d^n$

<value> $x^v = a^v$; $y^v = b^v / \exp(10, b^n)$; $w^v = c^v + d^v / \exp(10, d^n)$

$(x^n \le 15 \; ; \; 10^{-294} \le x^v \le 10^{+322})$

3.6) <real number with exponent> x ::= <real number without exponent> a 'E'
<integer> b

<value> $x^v = a^v \exp(10, b^v)$

$(a^n \le 15 \; ; 10^{-294} \le x^v \le 10^{+322})$

3.7) <unsigned real number> x ::= <real number without exponent> a ;
y ::= <real number with exponent> b

<value> $x^v = a^v$; $y^v = b^v$

3.8) <real number> x ::= <unsigned real number> a ; y ::= '+' <unsigned real number> b ; w ::= '-' <unsigned real number> c

<value> $x^v = a^v$; $y^v = b^v$; $w^v = -c^v$

3.9) <double precision number> x ::= <real number without exponent> a 'D'
<integer> b

<value> $x^v = a^v \exp(10, b^v)$

$(15 \le a^n \le 29 \; , \; 10^{-294} \le x^v \le 10^{+322})$

3.10) <complex constant> x ::= '(' <real number> a ',' <real number> b ')'

<value> $x^v = a^v + i b^v$

3.11) <Hollerith constant> x ::= <unsigned decimal integer> a 'H'
<character string> b

<value>^{v}

<length>^{n} $(a^v \le 10, b^n \le 10, a^v = b^n)$

3.12) <character string> x ::= <character> a ; y ::= <character string> b
<character> c
<length>^{n} ($x^n \leq 10$)

3.13) <logical constant> x ::= '.TRUE.' ; y ::= '.FALSE.'
<value> x^v = true ; y^v = false

3.14) <constant> u ::= <decimal integer> a ; v ::= <octal integer> b ;
w ::= <real number> c ; x ::= <double precision number> d ;
y ::= <complex number> e ; z ::= <Hollerith constant> f ;
t ::= <logical constant> g
<value> $u^v = a^v$; $v^v = b^v$; $w^v = c^v$; $x^v = d^v$; $y^v = e^v$; $z^v = f^v$; $t^v = g^v$

4. EXPRESSIONS (mixed mode is considered)

4.1) <+|-> p ::= '+' ; m ::= '-'
<binary operator> p^u = ident ; m^u = neg
<binary operator> p^b = plus ; m^b = difference

4.2) <*|/> p ::= '*' ; m ::= '/'
<binary operator> p^b = times ; m^b = divide.

4.3) <relational operator> u ::= '.EQ.' ; v ::= '.NE.' ; w ::= '.GT.' ; x ::= '.GE.' ;
y ::= '.LT.' ; z ::= '.LE.'
<binary operator> u^b = equal ; v^b = notequal ; w^b = greater ;
x^b = greaterorequal ; y^b = less ; z^b = less or equal

4.4) <logical operator> x ::= '.AND.' ; y ::= '.OR.' ~~; w ::= '.NOT.'~~
<binary operator> x^b = and ; y^b = or
~~<unary operator> w^u = negate.~~

4.5) <primary> x ::= <real number> a ; y ::= <integer number> b ;
w ::= <function name> c ; z ::= '(' <arithmatic expression> d ')'
<value> $x^v(S) = a^v(S)$; $y^v(S) = b^v(S)$; $w^v(S) = c^v(S)$; $z^v(S) = d^v(S)$
<side effect> $x^s(S) = S$; $y^s(S) = S$; $w^s(S) = c^s(S)$; $z^s(S) = d^s(S)$
<type function> ^{y}
<type> x^t = real ; y^t = integer ; $w^t = w^y(c)$; $z^t = z^y(d)$

4.6) <factor> x ::= <primary> a ; y ::= <factor> b '**' <primary> c
<value> $x^v(S) = a^v(S)$; $y^v(S) = exp(b^v(S), b^v(c^s(S)))$
<side effect> $x^s(S) = a^s(S)$; $y^s(S) = c^s(b^s(S))$
<type function> ^{y}
$a^y = x^y$; $b^y = y^y$; $z^y = y^y$

4.7) <term> x ::= <factor> a ; y ::= <term> b <*|/> c <primary> d
<value> $x^v(S) = a^v(S)$; $y^v(S) = c^v(b^v(S), d^v(b^s(S)))$
<side effect> $x^s(S) = a^s(S)$; $y^s(S) = d^s(b^s(S))$
<type function> ^{y}
$a^y = x^y$; $b^y = y^y$; $d^y = y^y$
<type> x^t = ctype(b^t, d^t) = if b^t higher precedence than d^t then b^t else d^t
(for order of dominance of type precedence, see Cal Run Fortran manual, pp 3-4)

4.8) <arithmatic expression> x ::= <term> a ; y ::= <+|-> b <term> c ;
w ::= <arithmatic expression> d <+|-> e <term> f
<value> $x^v(S) = a^v(S)$; $y^v(S) = b^u(c(S))$; $w^v(S) = e^b(d^v(S), f^v(d^s(S)))$
<side effect> $x^s(S) = a^s(S)$; $y^s(S) = c^s(S)$; $w^s(S) = f^s(d^s(S))$
<type function> ^{y}
$a^y = x^y$; $c^y = y^y$; $d^y = w^y$; $f^y = w^y$
<type> $x^t = a^t$; $y^t = c^t$; w^t = ctype(d^t, f^t)

4.9) <relational expression> x ::= <arithmetic expression> a <relational operator> b
<arithmetic expression> c

<value> $x^v(s) = b^b(a^v(s), c^v(a^s(s)))$

<side effect> $x^s(s) = c^s(a^s(s))$

<type> $x^t = ctype(a^t, c^t)$

4.10) <logical factor> x ::= <logical variable> a ; y ::= <logical constant> b ;
w ::= <relational expression> c ; z ::= '(' <logical expr> d ')'

<value> $x^v(s) = a^v(s)$; $y^v(s) = b^v(s)$; $w^v(s) = c^v(s)$; $z^v(s) = d^v(s)$

<side effect> $x^s(s) = s$; $y^s(s) = s$; $w^s(s) = c^s(s)$; $z^s(s) = d^s(s)$

<type> $x^t(s)$ = true or false

4.11) <logical term> x ::= <logical factor> a ; y ::= 'NOT.' <logical factor> b ;

<value> $x^v(s) = a^v(s)$; $y^v(s) = \overline{b^v}(s)$ ($\overline{b^v}$ means negation of b^v)

<side effect> $x^s(s) = a^s(s)$; $y^s(s) = b^s(s)$

4.12) <logical expression> x ::= <logical term> a <logical operator> b
<logical term> c ; y ::= <logical expression> d
<logical operator> e <logical term> f

<value> $x^v(s) = b^b(a^v(s), c^v(a^s(s)))$; $y^v(s) = e^b(d^v(s), f^v(d^s(s)))$

<side effect> $x^s(s) = c^s(a^s(s))$; $y^s(s) = f^s(d^s(s))$

4.13) <masking factor> x ::= <variable> a ; y ::= <constant> b ; w ::=
<expression> c ; z ::= '(' <masking expression> d ')'

<value> $x^v(s) = a^v(s)$; $y^v(s) = b^v(s)$; $w^v(s) = c^v(s)$; $z^v(s) = d^v(s)$

<side effect> $x^s(s) = s$; $y^s(s) = s$; $w^s(s) = c^s(s)$; $z^s(s) = d^s(s)$

<type function> ^{p}

$x^t = x^p(a)$; $y^t = b^t$; $w^t = w^p(c)$; $z^t = z^p(d)$

4.14) <masking term> x ::= <masking factor> a ; y ::= 'NOT.' <masking factor> b

<value> $x^v(s) = a^v(s)$; $y^v(s) = \overline{b^v}(s)$ (negation here is 1's C)

<side effect> $x^s(s) = a^s(s)$; $y^s(s) = b^s(s)$

<type function> ^{p}

$a^p = x^p$; $b^p = y^p$

4.15) <masking expression> x ::= <masking term> a <logical operator> b

<masking term> c ; y ::= <masking expression> d

<logical operator> e <masking term> f

<value> $x^v(s) = b^b(a^v(s), c^v(a^s(s)))$; $y^v(s) = e^b(d^v(s), f^v(d^s(s)))$

<side effect> $x^s(s) = c^s(a^s(s))$; $y^s(s) = f^s(d^s(s))$

<type function> ^{p}

$a^p = x^p$; $c^p = v^p$; $d^p = y^p$; $f^p = y^p$

4.16) <expression> x ::= <arithmetic expression> a ; y ::= <masking expression> b ;

w ::= <logical expression> c ; z ::= <relational expression> d

<type> $x^t(s) = a^t(s)$; $y^t(s) = b^t(s)$; $w^t(s) = c^t(s)$; $z^t(s) = d^t(s)$

{DECLARATIONS}

(SYNTAX & SEMANTICS)

June 13, 1973

5. DECLARATION

5.0) <subscript bound list> x ::= <integer> i ; y ::= <integer> j ',' <integer> k ;
z ::= <integer> l ',' <integer> m ',' <integer> n

<set of n-tuples> $x^u = \{k: 1 \le k \le i^v\}$; $y^u = \{k: 1 \le k \le j^v\} \times \{k: 1 \le k \le k^v\}$

$z^u = \{k: 1 \le k \le l^v\} \times \{k: 1 \le k \le m^v\} \times \{k: 1 \le k \le n^v\}$

<total no. of elements> $x^n = i^v$; $y^n = j^v \cdot k^v$; $z^n = l^v \cdot m^v \cdot n^v$

5.1) Type Declaration

5.1.1) <type> r ::= 'COMPLEX' ; s ::= 'DOUBLE PRECISION' | 'DOUBLE' ;
t ::= 'REAL' ; u ::= 'INTEGER' ; v ::= 'LOGICAL'

<type set> r^t = complex ; s^t = double precision ;
t^t = real ; u^t = integer ; v^t = boolean

5.1.2) <type segment> x ::= <simple variable> a ; y ::= <simple variable> b '(' <subscript bound list> c ')'

<type set> ^{t}
$a^t = x^t$; $b^t = y^t$
<set of n-tuples> $x^u = c^u$
<type function> $x^z = \{a, x^t\}$; $y^z = \{((b,k), y^t): k \in c^u\}$

5.1.3) <type list> x ::= <type segment> a ; y ::= <type list> b ',' <type segment> c

<type set> ^{t}
$a^t = x^t$; $b^t = y^t$
<type function> $x^z = a^z$; $y^z = b^z \cup c^z$

5.1.4) <type declaration> x ::= <type> a <type list> b

<type set> ^{t}
$b^t = x^t$
<type function> $x^z = b^z$

5.2) Dimension Declaration

5.2.1) <array segment> x::= <simple variable> a '(' <subscript bound list> b ')'
<type set> a^z = default ; $x^t = a^t$
<set of n-tuples> $x^u = b^u$
<type function> $x^z = \{((a,k), a^t) : k \in b^u\}$

5.2.2) <array list> x::= <array segment> a ; y::= <array list> b ',' <array segment> c
<type set> $x^t = a^t$
<type function> $x^z = a^z$; $y^z = b^z \cup c^z$

5.2.3) <dimension declaration> x::= 'DIMENSION' <array list> a
<type function> $x^z = a^z$

5.3) Common Declaration

5.3.1) <common segment> x::= <simple variable> a ; y::= <simple variable> b '(' <subscript bound list> c ')'
<set of n-tuples> $y^u = c^u$
<program name>^{p}
$a^p = x^p$; $b^p = y^p$
<block name>^{b}
$a^b = x^b$; $b^b = x^b$
<common variable index>^{i}
$a^i = x^i$; $b^i = y^i$
<local variable> $x^l = (x^p, a^i)$; $y^l = ((y^p, b^m) : a^i \le m \le a^i + c^m - 1)$
<global variable> $x^g = (x^b, a^i)$; $y^g = ((y^b, b^m) : a^i \le m \le a^i + c^m - 1)$
(if $x^p = \phi$ or $y^p = \phi$, local variable not defined ;
if $x^g = \phi$ or $y^g = \phi$, global variable not defined)

5.3.2) <common list> x::= <common segment> a ; y::= <common list> b 'g'
<common segment> c
<common variable index> ^{g}
$a^{g} = 1$; $c^{g} = b^{n} + 1$
<program name> ^{p}
$a^{p} = x^{p}$; $b^{p} = y^{p}$; $c^{p} = y^{r}$
<block name> ^{b}
$a^{b} = x^{b}$; $b^{b} = y^{b}$; $c^{b} = y^{b}$
<local variable> $x^{l} = a^{l}$; $y^{l} = b^{l} \cup c^{l}$
<global variable> $x^{g} = a^{g}$; $y^{g} = b^{g} \cup c^{g}$

5.3.3) <labelled common block> x::= '/' <alphanumeric identifier> a '/'
<common list> b ; y::= <labelled common block>
'/' <alphanumeric identifier> d '/'
<common list> e

<program name> ^{p}
$x^{p} = \phi$; $y^{p} = \phi$
<block name> ^{b}
$b^{b} = a^{b}$; $e^{b} = d^{b}$
<global variable> $x^{g} = b^{g}$; $y^{g} = c^{g} \cup e^{g}$

5.3.4) <blank common block> x::= <common list> a ; y::= '//'
<common list> b

<program name> ^{p}
$a^{p} = x^{p}$; $b^{p} = y^{p}$
<block name>
$x^{p} = \phi$; $y^{r} = \phi$
<local variable> $x^{l} = a^{l}$; $y^{l} = b^{l}$

5.3.5) <common block> x::= <blank common block> a; y::= <labelled common block> b

<program name>^{p}
$a^{p} = x^{p}$
<block name>^{b} $y^{b} = b^{b}$
<local variable> $x^{l} = a^{l}$
<global variable> $y^{g} = b^{g}$

5.3.6) <common declaration> x::= 'COMMON' <common block> a

<program name>^{p}
$a^{p} = x^{p}$
<block name>^{b} $x^{b} = a^{b}$
<local variable> $x^{l} = a^{l}$
<global variable> $x^{g} = a^{g}$

5.4) Data Declaration

5.4.1) <list of values> w::= <constant> a ; x::= <integer> i '*' <constant> b;
y::= <list of values> u ',' <constant> c;
z::= <list of values> v ',' <integer> j '*' <constant> d

<length> $w^{n} = 1$; $x^{n} = i^{v}$; $y^{n} = u^{n} + 1$; $z^{n} = v^{n} + j^{v}$

<value function> $w^{v} = \{(1, a^{v})\}$; $x^{v} = \{(k, b^{v}) : 1 \le k \le i^{v}\}$;
$y^{v} = u^{v} \cup \{(y^{n}, c^{v})\}$; $z^{v} = v^{v} \cup \{(k, d^{v}) : v^{n} \le k \le z^{n}\}$

5.4.2) <list of variable> w::= <alphanumeric identifier> c ; x::= '('
<variable> a ',' <alphanumeric identifier> i '='
<integer> s ',' <integer> e ')'; y::= <list of variable> u ',' <alphanumeric identifier> d ;
z::= <list of variable> v ',' '(' <variable> b ','
<alphanumeric identifier> j '=' <integer> t
',' <integer> f ')'

<length> $w^n = 1$; $x^m = e^v - s^v + 1$; $y^n = u^n + 1$;
$z^n = v^n + f^v - t^v + 1$
$(s \le e)$
$(t \le f)$
<list of variable> ^{v}
<partial initial statevector> $w^p = \{(c, w^v(1))\}$;
$x^p = \{(a^l(S_k), x^v(k-s+1)) : s \le k \le e\}$, where $S_k(i) = k$;
$y^p = u^p \cup \{(d, y^v(y^n))\}$; $z^p = v^p \cup \{(b^l(S_k)$,
$z^v(v^n + k - t + 1)) : t \le k \le f\}$, where $S_k(j) = k$
$(s \le S(i) = S'(i) \le e \Rightarrow a^l(S) = a^l(S'))$
$(t \le S(j) = S'(j) \le f \Rightarrow b^l(S) = b^l(S'))$

STATEMENTS

(SYNTAX & SEMANTICS)

June 13, 1973

6. STATEMENTS (UNLABELLED)

6.0) <unlabelled statement> x ::= <assignment statement> a ; y ::= <control statement> b ;
z ::= <input-output statements> c ; w ::= <subroutine calling statement> d ;

<statement index> i
$a^i = x^i$; $b^i = y^i$; $c^i = z^i$; $d^i = w^i$
<label function> a
$a^a = x^a$; $b^a = y^a$; $c^a = z^a$, $d^a = w^a$
<effect> $x^e(s) = a^e(s)$; $y^e(s) = b^e(s)$; $z^e(s) = c^e(s)$; $w^e(s) = d^e(s)$
<exit index> $x^x(s) = a^x(s)$; $y^x(s) = b^x(s)$; $z^x(s) = c^x(s)$; $w^x(s) = d^x(s)$

6.1) Assignment statement (unlabelled)

6.1.1) <assignment statement> a ::= <variable> v '=' <expression> e
<effect> $a^e(s) = s'$, where $s'(v^e(s)) = apply(mtcf(v^t, e^t), e^v(v^s(s)))$, $s'(z) = s''(z)$ for $z \neq v^e(s)$, where $s'' = e^s(v^s(s))$
<exit index> $a^x(s) = a^i + 1$
(type conversion is considered in effect attribute)

6.2) Control statement (unlabelled)

6.2.1) <control statement> x ::= <IF statement> a ; y ::= <GO TO statement> b ;
w ::= <DO statement> c ; z ::= <STOP statement> d ;
u ::= <CONTINUE statement> e ; v ::= <END statement> f
<effect> $x^e(s) = a^e(s)$; $y^e(s) = b^e(s)$; $w^e(s) = c^e(s)$; $z^e(s) = d^e(s)$
$u^e(s) = e^e(s)$; $v^e(s) = f^e(s)$
<exit index> $x^x(s) = a^x(s)$; $y^x(s) = b^x(s)$; $w^x(s) = c^x(s)$;
$z^x(s) = d^x(s)$; $u^x(s) = e^x(s)$; $v^x(s) = f^x(s)$

6.2.2) <IF statement> f ::= 'IF('<arithmetic expression> e ')'
<statement number> x 'y' <statement number> y 'z'
<statement number> z ; g ::= 'IF('<Boolean logical
expression> b ')' <statement> t ; h ::= 'IF('
<logical expression> ~ ')' <statement number> q ';'
<statement number> n

<statement index> ^{i}
<label function> ^{a}
$t^i = g^i$
<effect> $f^e(S) = S$; $g^e(S) = b^v(S)$ if $b^v(S)$ then $t^e(S)$ else S
$h^e(S) = S$
<exit index> $f^x(S) =$ if $e^v(S) < 0$ then $f^a(x)$ else if $e^v(S) = 0$
then $f^a(y)$ else $f^a(z)$; $g^x(S) =$ if $b^v(S)$
then $t^x(S)$ else $q^a + 1$; $h^x(S) =$ if $b \wedge c^v(S)$
then $h^a(q)$ else $h^a(n)$

6.2.3) <GO TO statement> x ::= <unconditional GO TO statement> a ;
y ::= <ASSIGN statement> b ; w ::= <assigned
GO TO statement> c ; z ::= <computed GO TO
statement> d

<statement index> ^{i}
$a^i = x^i$; $b^i = y^i$; $c^i = w^i$; $d^i = z^i$
<effect> $x^e(S) = a^e(S)$; $y^e(S) = b^e(S)$; $w^e(S) = c^e(S)$; $z^e(S) = d^e(S)$
<exit index> $x^x(S) = a^x(S)$; $y^x(S) = b^x(S)$; $w^x(S) = c^x(S)$; $z^x(S) = d^x(S)$

6.2.4) <unconditional GO TO statement> g ::= 'GO TO' <statement
number> n

<statement index> ^{i}
<label function> ^{a}
<effect> $g^e(S) = S$
<exit index> $g^x(S) \equiv g^a(n)$

6.2.5) <ASSIGN statement> s ::= 'ASSIGN' <statement number> n
'TO' <alphanumeric identifier> i
<effect> $S^e(S) = S'$, where $S'(i) = n$, $S'(z) = S(z)$
for $z \neq i$
<exit index> $S^x(S) = S^i + 1$

6.2.6) <assigned GO TO statement> s ::= 'GO TO' <simple variable> v ',(' <label list> ')'
<effect> $S^e(S) = S$
<label function> a
<exit index> $S^x(S) = s^a(S(v))$

6.2.7) <computed GO TO statement> s ::= 'GO TO (' <label list> a '),' <simple variable>
<effect> $S^e(S) = S$
<label function> a
<exit index> $S^x(S) =$ if $1 \leq S(v) \leq a^n$ then $s^a(a^f(S(v)))$
else error

6.2.8) <STOP statement> s ::= 'STOP'
<effect> $S^e(S) = S$
<exit index> $S^x(S) = S^i + 1$

6.2.9) <CONTINUE statement> s ::= 'CONTINUE'
<effect> $S^e(S) = S$
<exit index> $S^x(S) = S^i + 1$

6.2.10) <END statement> s ::= 'END'
<effect> $S^e(S) = S$
<exit index> $S^x(S) =$ end

6.2.11) <statement> x::= <unlabelled statement> u; y::= <statement identifier> a' <statement> z

<statement index> i

$u^i = x^i$; $z^i = y^i$

<local label function> $u^b = \phi$; $y^b = \{(a, y^i)\}$

<exit index> $x^x(S) = u^x(S)$; $y^x(S) = z^x(S)$

6.3) Dϕ statement

6.3.1) <Dϕ statement> x::= 'Dϕ' <statement identifier> s <variable> v '=' <two or three integers>

6.3.2) <two or three integers>::= <unsigned integer> ',' <unsigned integer> | <unsigned integer> ',' <unsigned integer> ',' <unsigned integer>

6.3.3) <Dϕ ~~statement~~ section> x::= <statement> s ; y::= <section> z ';' <statement> t

<number of statements> $x^n = s^n$; $y^n = z^n + t^n$

<statement index> i

$s^i = x^i$; $z^i = y^i$; $t^i = y^i + z^n$

<single step function> $x^p((S,1)) = (s^e(S), s^x(S))$;

$y^p((S,k)) = z^p((S,k))$ for $1 \le k \le z^n$;

$y^p((S,k)) = (t^e(S), t^x(S))$

6.3.4) <Dϕ loop section> x::= <Dϕ statement> a <Dϕ section> b

6.4) Input-Output Statement

6.4.1) <READ Statement> χ ::= 'READ' <statement identifier> i ',' <variable> v;
η ::= <READ statement> z ',' <variable> w

<effect> $\chi^e(S) = S'$, where if $S(\underline{reader}) = (x_1, x_2, \ldots, x_n)$ then $S'(\underline{reader}) = (x_2, \ldots, x_n)$, $S'(v) = x_1$, and $S'(z) = S(z)$ for $z \neq v$, reader ; $\eta^e(S) = S'$, where if $z^e(S) = S''$ and $S''(\underline{reader}) = (x_1, x_2, \ldots, x_n)$ then $S'(\underline{reader}) = (x_2, \ldots, x_n)$, $S'(w) = x_1$, and $S'(z) = S''(z)$ for $z \neq w$, reader

<exit index> $\nu^\chi(S) = \chi^i + 1$

6.4.2) <PRINT statement> χ ::= 'PRINT' <statement identifier> i ',' <variable> v ; η ::= <PRINT statement> z ',' <variable> w

<effect> $\chi^e(S) = S'$, where if $S(\underline{printer}) = (x_1, \ldots, x_n)$ then $S'(\underline{printer}) = (S(v), x_1, \ldots, x_n)$, $S'(z) = S(z)$ for $z \neq$ printer ; $\eta^e(S) = S'$, where if $z^e(S) = S''$ and $S''(\underline{printer}) = (x_1, \ldots, x_n)$ then $S'(\underline{printer}) = (S(w), x_1, \ldots, x_n)$, $S'(z) = S''(z)$ for $z \neq$ printer

<exit index> $\chi^x(S) = \chi^i + 1$

6.4.3) <READ Statement> π ::= 'READ (' <file name> m ',' <statement identifier> i ')' <variable> x

<effect> $\pi^e(S) = S'$, where $S'(m^p) = S(m^p) + 1$, $S'(x) = S(m^m(S(m^p) + 1))$, $S'(z) = S(z)$ for $z \neq m^p, x$

<exit index> $\pi^x(S) = \pi^i + 1$

6.4.4) <WRITE Statement>$_w$ ::= 'WRITE(' <file name>$_n$ ',' <statement identifier>$_i$ ')' <variable x>

<effect> $w^e(S) = S'$, where $S'(n^p) = S(n^p) + 1$,
$S'(n^m(S(n^p)+1)) = S(x)$, $S'(z) = S(z)$ for
$z \neq n^m(S(n^p)+1), x$

<exit index> $w^x(S) = w^i + 1$

6.5) Subroutine Calling Statement

6.5.1) <Subroutine Calling Statement>$_x$ ::= 'CALL' <subroutine name>$_s$ '(' <actual parameter list>$_a$ ')'

<effect> $x^e(S) = s^e(S)$

<side effect> $x^s(S) = s^s(S)$

<exit index> $x^x(S) =$ if $s^x(S) \neq$ normal then $s^x(s^e(S))$
else $x^i + 1$

PROGRAMS

(SYNTAX & SEMANTICS)

June 13, 1973

PROGRAMS

(SYNTAX & SEMANTICS)

June 13, 1973

7. PROGRAMS

7.0) <declaration> x ::= <type declaration> a ; y ::= <dimension declaration> b;
w ::= <common declaration> c ; z ::= <data declaration> d
<local environment> $x^z = a^z$; $y^z = b^z$; $w^z = c^z$; $z^z = d^z$

7.1) Programs

7.1.1) <set of declarations> x ::= <declaration> d ; y ::= <set of declaration> z^1 ';' <declaration> e
<local environment> $x^z = d^z$; $y^z = z^z \cup e^z$

7.1.2) <set of statements> x ::= <statement> s ; y ::= <set of statements> z ';' <statement> t
<environment> ^{y}
$s^y = x^y$; $z^y = y^y$; $t^y = y^y$

7.1.3) <~~main~~ program body> p ::= <set of declarations> a ';' <set of statements> b
<local environment> ^{z}
<environment> $'^y$
$b'^y = a^z$

7.1.4) <main program> s ::= 'PROGRAM' <main program name> n ';'
<program body> b ';' <END statement> ;
x ::= <main program name> p^1 ';' <program body> q^1 ';' <END statement>
<local environment> $s^z = b^z \cup \{((n, \underline{\text{effect}}), b^q)^y_z, ((n, \underline{\text{exit}}), b^x)\}$; $x^z = p^z \cup \{((p, \underline{\text{effect}}), q'^y_z, ((p, \underline{\text{exit}}), q^x)\}$
<environment> $b'^y = b^z$; $q'^y = q^z$

7.1.5) <subroutine> s ::= 'SUBROUTINE' <subroutine name> n ';'
<program body> b ';' <END statement>
<local environment> $s^z = b^z \cup \{((n, \text{effect}), b^e), ((n, \text{exit}), b^x)\}$
<environment> ^{y} $b^y = b^z$

7.1.6) <function subprogram> s ::= 'FUNCTION' <function name> n ';'
<program body> b ';' <END statement> ;
x ::= <type> m 'FUNCTION' <function name> p
';' <program body> q ';' <END statement>
<local environment> $s^z = b^z \cup \{((n, \text{effect}), b^e), ((n, \text{exit}), b^x),$
$((n, \text{type}), \text{real})\}$; $x^z = q^z \cup$
$\{((p, \text{effect}), q^e), ((p, \text{exit}), q^x),$
$((p, \text{type}), m^t)\}$
<environment> ^{y}
$b^y = b^z$; $q^y = q^z$

7.1.7) <program> x ::= <main program> m ; y ::= <function
subprogram> f ; z ::= <subroutine> s
<local environment> $x^z = m^z$; $y^z = f^z$; $z^z = s^z$
<environment> ^{y}
$m^y = x^y$; $f^y = y^y$; $s^y = z^y$

7.1.8) <collection of programs> x ::= <program> p ; y ::= <collection
of programs> z <program> q
<local environment> $x^z = p^z$; $y^z = z^z \cup q^z$
<environment> ^{y}
$p^y = x^y$; $z^y = y^y$; $q^y = y^y$

7.1.9) <complete collection of programs> ::= <collection of programs>
<end marker>
<local environment> ^{z}
<environment> ^{y}
$c^y = c^z$

7.2) Entry & External Statements of programs

7.2.1) <ENTRY statement> x ::= 'ENTRY' <entry name> a
<generalized effect> β
<label function> a^a
<name function> $x^m = \{(a, f)\}$ where $f(S) = S'$ for $(S', k') = x^\beta((S, x^a(a)))$
<global variable> $a^{\beta b} = (\$GLOBAL, 'a')$

7.2.2) <EXTERNAL Statement> x ::= 'EXTERNAL' <variable> v
<global variable> $v^{\beta b} = (\$GLOBAL, v)$

C H A P T E R S E V E N

A D E S C R I P T I O N O F L I S P

B Y

N A N C Y H . M c D O N A L D

LISP

LISP is a list-processing language which bears little resemblance to the better known algebraic languages such as FORTRAN or ALGOL. However, many of the ideas presented by Professor Maurer in "Syntax and Semantics of Programming Languages" are directly applicable. Due to LISP's mathematical nature and recursive capabilities, its simple syntax very often belies the semantic complexity.

In an attempt to structure this report in a similar fashion to Professor Maurer's book, I have divided the language into four categories: Constants and Primaries; Expressions; Statements; and Program Feature. Under the heading of Constants and Primaries, I have placed the definitions of octal and decimal numbers, T, NIL, atoms, and the basic S-expression format. The unit entitled Expressions contains lambda-expressions and forms, while the Statement section includes what I have dubbed "lambda-statements" (encompassing the bindings), SET, SETQ, CSET, and CSETQ functions to correspond to algebraic assignment, conditional, and transfer statements. Although there is no reason one should need a sequential program with transfer statements, given recursion and the conditional expression, I have included the program feature to correspond to the concept of "program" as it is generally defined in algebraic languages.

Constants and Primaries

In LISP, constants are entities which contain one value for an entire program. Unlike FORTRAN where a constant may be changed if used as the argument of a subprogram call, LISP constants--numberic atoms, literal atoms, and free variables have values equivalent to their 'names' throughout the program.

I chose to include the notion of atoms, s-expressions and free vatiable (identifier) here because of their primary nature and usage. All LISP expressions are combinations of s-expressions whigh are combinations of atoms.

Constants and Primaries

<octal digit> a::= '0'; b::= '1'; c::= '2'; d::= '3';
e::= '4'; f::= '5'; g::= '6'; h::= '7'

<value> $a^v = 0$, $b^v = 1$, $c^v = 2$, $d^v = 3$, $e^v = 4$,
$f^v = 5$; $g^v = 6$, $h^v = 7$
[all lengths are 1]

<digit for decimal digit> x::= <octal digit> d;
y::= '8'; z::= '9'

<value> $x^v = d^v$; $y^v = 8$, $z^v = 9$
[lengths are all 1]

<octal integer> x::= <octal digit> d 'Q';
y::= <octal digit> e <octal integer> z

<value> $x^v = d^v$, $y^v = z^v + e^v * \exp(8, z^n)$
<length> $x^n = 1$; $y^n = 1 + z^n$
[you might consider adding 1 to each length attribute to include the 'Q']

<decimal integer> x::= <digit> d; y::= <digit> e
<decimal integer> z

<value> $x^v = d^v$; $y^v = z^v + e^v * \exp(10, z^n)$

<unsigned integer> x::= <octal integer> o; y::= <decimal integer> d

<value> $x^v = o^v$; $y^v = d^v$
<length> $x^n = o^n$; $y^n = d^n$

<integer> x::= <unsigned integer> u; y::= '+' <unsigned integer> v; z::= '-' <unsigned integer> w

<value> $x^v = u^v$; $y^v = v^v$; $z^v = -w^v$
<length> $x^n = u^n$; $y^n = v^n$; $z^n = w^n$
[v^n and w^n might be increased by 1 to include the sign]

<integer number> x::= <integer> a; y::= <decimal integer> b 'E' <unsigned integer> c;
z::= <octal integer> d 'Q' <unsigned integer> e

<value> $x^v = a^v$; $y^v = b^v * exp(10, e^v)$;
$z^v = d^v * exp(8, e^v)$

<ufpn> ≡ <unsigned floating point number> x ::= <decimal integer> a '.'; y ::= <decimal integer> b '.' <decimal integer> c; z ::= '.' <decimal integer> d

<value> $x^v = a^v$; ⋮ $y^v = b^v + c^v / \exp(10, c^n)$; $z^v = d^v / \exp(10, d^n)$

<length> $x^n = a^n$, $y^n = b^n + c^n$; $z^n = d^n$

[y^n and z^n might be increased by 1 to include the decimal point]

<sfpn> ≡ <signed floating point number> x ::= '+' <ufpn> a; y ::= '-' <ufpn> b

<value> $x^v = a^v$; $y^v = -b^v$

<length> $x^n = a^n$; $y^n = b^n$ [or add 1 for sign]

<floating point number> x ::= <sfpn> a 'E' <decimal integer> g; y ::= <sfpn> b 'E+' <decimal integer> h; z ::= <sfpn> c 'E-' <decimal integer> i

<value> $x^v = a^v * \exp(10, g^v)$; $y^v = b^v * \exp(10, h^v)$; $z^v = c^v * \exp(10, -i^v)$

<length> as above

<number>x::= <integer number>a; y::= <floating point number>b

<value> $x^v = a^v$; $y^v = b^v$

<length> $x^n = a^n$; $y^n = b^n$

Primary

<literal atom> a ::= <letter> l ; b ::= <literal atom> e
<alphameric> m
<length> $a^n = 1$; $b^n = m^n + c^n$
<letter> l ::= 'a' | 'b' | 'c' | ... 'z'

<alphameric> a ::= <letter> l ; b ::= <decimal digit> d

<numeric atom> n ::= <number>

<atom> ::= <numeric atom> | <literal atom>

<s-expression> s ::= <atom> a ; t ::= '(' <atom> b '.' <atom> c ')'
u ::= '(' <s-expression> d '.' <s-expression> e ')'

<list> l ::= '(' <atom> a ')' ; m ::= '(' <atom> b
<list> n ')'

<constant> c ::= <number> n ; d ::= 'QUOTE'
<literal atom> l ; e ::= <free variable> f

c^v = value cell (n)
d^v = value cell (l) = 'l'
e^v = value cell (f)

To set constants use one of 2 functions-

<cset fcn> c ::= '(CSET' <variable> v
<variable> w ')'
$c^{e(s)}$ = $v^v = w^v$
where v^v must be a literal atom

<esetq fcn> c ::= '(CSETQ' <variable> v
<variable> w
$c^{e(s)}$ = value cell (v) = w^v

variable

<free variable> v ::= <literal atom> a

<length> v^n ::= a^n

Type is not specified by the variable in LISP. Any type may be stored in any variable. The only time type is checked is upon executing a function which expects a particular type. Finding an unexpected type will cause an error and escape condition.

<bound variable> v ::= <literal atom> a

<value> v^v = value bound to a

Expressions

'Expression' is somewhat of a misnomer due to the fact that everything could be named an expression (and generally is) in LISP. I have chosen to categorize the LAMBDA expression and the concept of forms as the class 'expressions'.

The body of the LAMBDA expression is not unlike an algebraic formula in that it describes how the variables are to be combined. The varlist merely defines which are the variables (in the FORTRAN sense). Thus, in FORTRAN an expression such as A*B+C might look like:

(LAMBDA (A B C) (PLUS (TIMES A B) C)) in LISP. Similarly, and since LISP functions are built out of LAMBDA expressions, I chose to include the idea of forms as forms are funcion-like entities.

"Expressions"

<lambda-expression> $l ::=$ 'LAMBDA' <var-list> v <body> t $::=$

<effect> $l^e = \{$associating s to $t\}$ or $t = s^a(s^e)$
$\{$the varlist names the variables of the body$\}$ $s; \xrightarrow{a} t$

To maintain Lisp's description -

<varlist> $v ::=$ '(' <literal atom> l v; $w ::=$ '(' <literal atom> m <varlist> x ')'

The var-list sets up an environment as does the DEFINE pseudo function described later.

<body> $f ::=$ <elementary form> e; $g ::=$ <composed form> c;
$h ::=$ <special form> s; $i ::=$ <prog feature>

Expr

<elementary form> e::= <variable>v; f::= <constant>c;
g::= <simple form>s

<value> $e^v(s) = v(s)$; $f^v(s) = c$; $g^v(s) = s^v(s^s)$

<simple form> s::= <fname>f<one or more variables>p

<value> s^v = apply (f, p^v).
where apply(g,t) means apply the function g to the arguments of t.

<name>^{n}
$f^n(S) = s^n(S)$
The name is inherited from the system

<fname> f::= "internal function name"
(built in function)

where "internal function name" are such as:
CAR, CDR, CONS ...

<composed form> f ::= <frame> q <args> r

<value> $f^v(s) = apply(q, r^v(r^s(s)))$

<args> r ::= <constant> c; s ::= <variable> d; t ::= <simple form> e; u ::= <composed form> f; ...

<value> $r^v = c^v$; $s^v = d^v$; $t^v = e^v(e^s)$; $u^v = f^v(f^s)$

<special forms> s::= <fname>f <args>a ;
t::= <fname>g <args>b

s^v = apply (f, $a^v(s)$) where a^n (represents the number of args) is unlimited

t^v = apply(f, b) - i.e. don't evaluate a.

<arg1>a::= ___ ; b::= <args>z
This is to represent 0 or more arguments.

<expression> e::= '(' <lambda-expr> || <form> ')'

<value> e^v = l^e | f^v

Statements

In order to approximate the type of ideas used in the Maurer book, I have isolated some commonly used expressions which have either an effect on the state vector or cause transfers of control(also changing the state vector). The LAMBDA statement causes a binding or 'assignment' of values to take place as does the 'assignment-like' functions SET, SETQ, CSET, CSETQ. The I/O statements have effects on the parts of the state vector representing the reader and printer. I found it necessary to restore the definition of statement indices so that I could allow for recursion and such multi-statements as conditional expressions. It was decided to include conditional expressions in the statement section because they themselves might contain various statements. And, transfer statements must be included as the only means for branching (in the usual sense of the word), although they are only permitted within a PROG feature.

'STATEMENTS'

<lambda stmt> l ::= <lambda-expression>e <s-expression>S

<effect> $l^e(s) = e^e(s(S))$

In other words, here we take the varlist of e which has mapped onto the body of e and also map the parts of S into the same area of e.

varlist of e, $\xrightarrow{\text{onto}}$ S; $\xrightarrow{\text{onto}}$ body of e^{evaluate}

In ordered pair notation:

((varlist of e, S), body of e)

assignment-like statements.

Global

<cset stmt> c::='CSET' <variable>v <variable>w

<effect> $c^e(S) = S'$ where $S'(v^v) = w^v(S)$, $S'(z) = S''(z)$ for $z \neq v^v$ where $S'' = w^s(S)$

(v^v must be literal atom)

<csetq stmt) c::='CSETQ' <variable>v <variable>w

<effect) $c^e(S) = S'$ where $S'(v) = w^v(S)$, $S'(z) = S''(z)$ for $z \neq v$ where $S'' = w^s(S)$

Local

<set stmt>s::='SET' <variable>v <variable>w

<effect> $s^e(S) = S'$ where $S'(v^v) = w^v(S)$, $S'(z) = S''(z)$ for $z \neq v^v$ where $S'' = w^s(S)$

(v^v must be a literal atom and prog variable)

<setq stmt) s::='SETQ' <variable) v <variable)w

<effect) $s^e(S) = S'$ where $S'(v) = w^v(S)$, $S'(z) = S''(z)$ for $z \neq v$ where $S'' = w^s(S)$

(v must be a prog variable)

I/O

<read statement> r ::= '(READ ( ))'

<effect> $r^e(S) = S'$ where if S(reader) = $(x_1, x_2, \ldots, x_n)$ then S'(reader) = $(x_2, x_3, \ldots, x_n)$.
$S'(z) \neq S(z)$ for z = reader

<print statement> p ::= '(PRINT (' <s-expr> s
'))'

<effect> $p^e(s) = S'$ where if S(printer) =
$(x_1, x_2 \ldots x_n)$ then S'(printer) = $(S(s), x_1, x_2 \ldots x_n)$
$S'(z) \neq S(z)$ for $z \neq$ printer

I/O

<read statement> r ::= '(READ ( ))'

<effect> $r^e(S) = S'$ where if S(reader) = $(x_1, x_2, \ldots, x_n)$ then S'(reader) = $(x_2, x_3, \ldots, x_n)$.
$S'(z) \neq S(z)$ for z = reader

<print statement> p ::= '(PRINT (' <s-expr> ')'

<effect> $p^e(s) = S'$ where if S(printer) = $(x_1, x_2 \ldots x_n)$ then S'(printer) = $(S(s), x_1, x_2 \ldots x_n)$
$S'(z) \neq S(z)$ for $z \neq$ printer

I wish to redefine statement indices to reflect the complexity of multi-statements (like conditionals and lambda) and recursion

A statement index will be an ordered pair (r, i) where r=1 will denote the possibility for the statement having recursive levels (i.e. it is used in a recursive section) or r=0 denoting a simple non-recursive statement; i will be a pair (c, s) where c = actual statement index for a multi-stmt and $1 \leq s \leq n$ where n is the number of <statement> within the multi-stmt.

e.g.

statement index	Statement
(0,1)	(SETQ A 1)
(0,2)	(SETQ B 2)
(0,(3,1))	(COND ((ATOM C)(GO LAB2))
(0,(3,2))	(T (SETQ D (CAR C))))
(1,(4,1))	(LABEL FIND (LAMBDA (X)
(1,(4,2))	(COND ((ATOM X) X)
(1,(4,3))	(T (FIND (CAR X))))))(D)

Note: obviously the recursive indicator will be inherited from the semantics of the statement itself.

conditional expressions

<cond exp> c ::= " COND " <pred-form> p

<effect> $c^e(S) = p^e(S)$

<exit index> $c^x(S) = p^x(S)$

<statement index>^{i} $p^i = c^i$

<pred-form> p ::= '(' <predicate fcn> f <statement> g ')' ; r ::= '(' <predicate fcn> h <statement> i ')' <pred-form> q

<statement index>^{i} $g^i = (p^i, 1)$, $i^i = (r^i, 1)$ $q^i = i^i_{+1}$

where i^i_{+1} means $(r^i, 1+1)$

<effect> p^e = if f = true then g^e

r^e = if h = true then i^e else q^e

<value> p^v = if f = true then g^v

r^v = if h = true then i^v else q^v

<label fcn>^{a}

<exit index> p^x = if f = true and g is a transfer statement then $p^a(g)$ else p^x_{+1}

r^x = if h = true and i is a transfer then $p^a(i)$ else q^x

<predicate function> p ::= "T"; q ::= "NIL"; r ::= <fname> f <args> a.

(value) p^v = true; q^v = false; r^v = apply(f, a^v)
where f may return the value
true or false only

transfer statement

<transfer statement) ::= 'GO' <label>'
<label function>^{a}
(exit index) $t^{x}_{\beta} = t^{a}(l)$
<effect> $\in \mathcal{E}^{e}_{\alpha\beta} = S$

Allowable only within PROG feature.

<unlabel-statement> s ::= '(' <conditional expression> c ')' ;
t ::= '(' <form> f ')' ; u ::= <transfer statement> g
<statement index>^{i} $c^i = s^i$; $f^i = t^i$; $g^i = u^i$

<label statement> x ::= <label> l <statement> s.
<statement index>^{i} $s^i = x^i$ where $x^i = (r, (c, l))$
<label function>^{a} $x^a(s) = l$

Program Feature

The program feature allows for sequential execution of statements as well as transfer of control to particular statements. This feature can be used to describe an entire LISP program or any part of a program. It need not be used at all, but I have included it in order to be able to include transfer and labelled statements.

The reader can note also that the definition of <section> applies to LISP programs based on recursion as well as based on a PROG. The prog-variable is a declaration of the local variables allowed to exist throughout the PROG and as such it sets up a local environment inherited by the section. (Lambda variables are identical in usage but limited to the single LAMBDA expression) A somewhat similar situation (though quite different usage) is shown [illegible] after the PROG - the pseudo function DEFINE actually sets up function names and their definitions to establish more of the environment to be inherited by the program in order for the functions to perform properly when invoked.

Program Feature

The program feature allows for sequential execution of statements as well as transfer of control to particular statements. This feature can be used to describe an entire LISP program or any part of a program. It need not be used at all, but I have included it in order to be able to include transfer and labelled statements.

The reader can note also that the definition of <section> applies to LISP programs based on recursion as well as based on a PROG. The prog-variable is a declaration of the local variables allowed to exist throughout the PROG and as such it sets up a local environment inherited by the section. (Lambda variables are identical in usage but limited to the single LAMBDA expression) A somewhat similar situation (though quite different usage) is shown after the PROG - the pseudo function DEFINE actually sets up function names and their definitions to establish more of the environment to be inherited by the program in order for the functions to perform properly when invoked.

program feature

<prog feature> p::= '(PROG' <prog-variables> v
<section> s
<statement index> i $s^i = p^i$
<local environment> $p^z = v^z$
$s^y = v^z$
<number of statements> $p^n = s^n$
<generalized effect> $p^g((S,k)) =$ if $p^i \le k \le p^{i+1} + s^n$
then $p^g(s^p((S,k)))$ else (S,k)
<effect> $p^e(S) = S'$ where $(S',k') = p^g((S,p^i))$
for some k'
<exit index> $p^x(S) = k'$, where $(S',k') = p^g((S,p^i))$ for
some S'
<section> s ::= <statement> a; t ::= <section> b
<statement> c.
<statement index> i $a^i = s^i$; $b^i = t^i$; $c^i = t^{i+} b^n$
<number of statements> $s^n = 1$; $t^n = b^n + 1$
<single step function> $s^p((S,1)) = (a^e(S), a^x(S))$;
$t^p((S,k)) = b^p((S,k))$ for $1 \le k \le b^n$;
$t^p((S,t^n)) = (c^e(S), c^x(S))$

Like Declarations (ot Chap 2)

pseudo functions

<define fcn> $d ::=$ ' DEFINE(' <fcn list>f')'

<effect> $d^e(s) = f^e(f^s(s))$

<local environment> $d^z = f^z \cup \{((\text{fname}, \underline{\text{effect}}), f^e), ((\text{fname}, \underline{\text{exit}}), f^x)\}$

<fcn list> $f ::=$ '('<fname> n <LAMBDA fcn> l ')'

$g ::=$ '('<fname> m <Lambda fcn> k ')'
<fcn list> h

<local environment> $f^z = (n \cdot l)^z$

$g^z = (m \cdot k)^z \cup h^z$

(where n·l means the combination of the fname and lambda fcn)

<exit index> $f^x = (r-1, (f^{i+1}, \underline{\text{anything}}))$

$g^x = (r-1, (g^{i+1}, \underline{\text{any}}))$

<statement index>^{i} $l^i = (r, (f^i, 1))$; $k^i = (r, (g^i, 1))$

$h^i = (r, (g^i, \Delta(g^{i+1})))$

BIBLIOGRAPHY

Maurer, W. D. The Programmer's Introduction to LISP. Macdonald London and Amevear Elsevier Inc., New York, 1972.

Maurer, W. D. Introduction to Programming Science-Part I: Syntax and Semantics of Programming Languages. Electronics Research Laboratory, Univ. of Calif., Berkeley, December, 1972.

McCarthy,J., et al. LISP 1.5 Programmer's Manual, 2nd Edition. MIT Press, Cambridge, 1965,

Weissman,C. LISP 1.5 Primer. Dickenson Press, Belmont, Calif., 1967.

C H A P T E R E I G H T

A D E S C R I P T I O N O F C O B O L

(as described in Stern and Stern's COBOL Programming)

B Y

S T E V E Z A L E W S K I

Page 1

<integer> x ::= <unsigned integer> u ; y ::= '-' <unsigned integer> v ; z ::= '+' <unsigned integer> w

<value> $x^v = u^v$; $y^v = -v^v$; $z^v = w^v$

<unsigned integer> x ::= <digit> u ; y ::= <unsigned integer> z <digit> v

<value> $x^v = u^v$; $y^v = 10 \times z^v + v^v$

<length> $x^n = (1, 0)$; $y^n = z^n + (1, 0)$

<type> x^t = integer ; y^t = integer

<decimal number> x ::= <integer> a '.' <unsigned integer> b ; y ::= '.' <unsigned integer> c ; z ::= '-' '.' <unsigned integer> d ; w ::= '+' '.' <unsigned integer> e

<value> $x^v = a^v + b^v \times exp(10, -b^n)$; $y^v = c^v \times exp(10, -c^n)$

$z^v = -d^v \times exp(10, -d^n)$; $w^v = e^v \times exp(10, -e^n)$

<type> x^t = real ; y^t = real ; z^t = real ; w^t = real

$(a^n + b^n \leq 18)$

<length> $x^n = (a^n, b^n)$; $y^n = (0, c^n)$; $z^n = (0, d^n]$

$w^n = (0, e^n)$

<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

<literal> x ::= <integer> a ; y ::= <decimal number> b

<value> $x^v = a^v$; $y^v = b^v$

<type> $x^t = a^t$; $y^t = b^t$

<length> $x^n = a^n$; $y^n = b^n$

<letter> ::= 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I'
| 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T'
| 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'

<let-dash> ::= <letter> | '-'

<variable> x ::= <letter> a ; y ::= <variable> b
<let-dash> c
<length> $x^n = 1$; $y^n = b^n + 1$ $(y^n \leq 30)$
<type> ^{t}

<Program> x ::= <Identification Division> <Environment
<Division> a <Data Division> b <Procedure
Division> c.
<effect> $x^e(s) = c^e(b^e(a^e(s)))$

<Procedure Division> x ::= 'Procedure Division.'
<p statement> a ; y ::= <Procedure Division> b
<p statement> c
<number of statements> $x^n = 1$; $y^n = b^n + 1$
<index> ^{i}
$a^i = 1$; $c^i = y^n$
<effect> $x^e(s) = a^e(s)$; $y^{e(s)} = c^e(b^e(s))$

<p statement> x ::= <unlabelled p statement> a ; y ::=
<variable> b <unlabelled p statement> c

< statement index > ^{i}

$a^i = x^i$; $c^i = y^i$

< label function > $a^a = \phi$; $y^a = \{(b, y^i)\}$

< unlabelled statement > x ::= < Imperative Statement > a ; y ::= < If statement > b ; z ::= < Authmetic GO TO statement > c

< statement index > ^{i}

$a^i = x^i$; $b^i = y^i$; $c^i = z^i$

< effect > $x^e(s) = a^e(s)$; $y^e(s) = b^e(s)$; $z^e(s) = c^e(s)$

< Imperative Statement > x1 ::= < Add Statement > a ;

x2 ::= < Subtract Statement > b ;

x3 ::= < Divide Statement > c ;

x4 ::= < Multiply Statement > d ;

x5 ::= < GO TO Statement > e ;

x6 ::= < Move Statement > f ;

x7 ::= < Write Statement > g ;

x8 ::= < Read Statement > h ;

x9 ::= < Open Statement > i ;

x10 ::= < Alter Statement > j ;

x11 ::= < Perform Statement > k

< statement index > ^{i}

$a^i = x1^i$; $b^i = x2^i$; $c^i = x3^i$; $d^i = x4^i$; $e^i = x5^i$; $f^i = x6^i$; $g^i = x7^i$; $h^i = x8^i$; $i^i = x9^i$; $j^i = x10^i$; $k^i = x11^i$

< effect > $x1^e(s) = a^e(s)$; $x2^e(s) = b^e(s)$; $x3^e(s) = c^e(s)$; $x4^e(s) = d^e(s)$; $x5^e(s) = e^e(s)$; $x6^e(s) = f^e(s)$; $x7^e(s) = g^e(s)$; $x8^e(s) = h^e(s)$; $x7^e(s) = i^e(s)$; $x10^e(s) = j^e(s)$; $x11^e(s) = k^e(s)$

<Identification Division>x ::= 'Identification Division.'
'Program-ID.' 'AUTHOR.' 'Installation.'
'Date-Written.' 'Date-Compiled.' 'Security.'
'Remarks.'

<Environment Division>x ::= <Configuration Section>a
<Input-Output Section>b
<effect> $x^e(s) = b^e(a^e(s))$

<Configuration Section>x ::= 'Configuration Section.'
'Source-Computer.' <variable> '.'
'Object-Computer.' <variable> '.'
<effect> $x^e(s) = S$

<Input-Output Section>x ::= 'Input-Output Section.'
'File-Control.' <select>a
<effect> $x^e(s) = a^e(s)$

<select>x ::= 'SELECT' <variable>a 'ASSIGN TO'
<figurative constant>b '.' ; y ::= <select>z 'SELECT'
<variable>c 'ASSIGN TO' <figurative constant>d
<effect> $x^e(s) = s'$ where $s'(a) = b$ and $s'(p) = S(p)$
for $p \neq a$; $y^e(s) = s'(z^e(s))$ where $s'(c) = d$
and $s'(p) = S(p)$ for $p \neq c$

<Data Division>x ::= 'DATA DIVISION.' 'FILE SECTION.'
<File Section>a ; y ::= 'DATA DIVISION.' 'FILE SECTION.'

<File Section> b 'Working-Storage Section.'

<Working Storage Section> c

<effect> $x^e(s) = a^e(s)$; $y^e(s) = c^e(b^e(s))$

<File Section> x ::= <FD> a ; y ::= <File Section> b <FD> c

<effect> $x^e(s) = a^e(s)$; $y^e(s) = c^e(b^e(s))$

<file environment> $x^f = a^f$; $y^f = b^f \cup c^f$

<record environment> $x^r = a^r$; $y^r = c^r$

<FD> x ::= 'FD' <variable> a '.' 'Recording Mode is'
<R mode> b '.' 'Label records are' <LMODE> c '.'
'Record contains' <integer> d 'characters.'
'Data record is' <variables> e '.' <R-section> f

<file environment> $x^f = \{((a, \text{recording mode}), b^v),$
$((a, \text{label records}), c^v), ((a, \text{record contains}), d^v),$
$(((a, \text{data record}), t) ; t \in e^v)$

<file name> ^{fn}
$f^{fn} = a$

<effect> $x^e(s) = s$

<record environment> $x^r = f^r$

<variables> x ::= <variable> a ; y ::= <variables> b
<variable> c

<value> $x^v = a$; $y^v = b \cup c$

<R mode> x ::= 'F' ; y ::= 'V' ; z ::= 'U'

<value> x^v = fixed ; y^v = variable ; $z^v = \emptyset$

<L mode> x ::= 'OMITTED' ; y ::= 'STANDARD'
<value> x^v = omitted ; y^v = standard

<IS-ARE> x ::= 'IS' ; y ::= 'ARE'

<R-section> x ::= '01' <variable> a '.' <R-subsection> b
<file record name> ^{frn}
$b^{frn} = (x^{fn}, a)$
<record environment> $x^r = b^r$

<R-subsection> x ::= <integer> a <variable> b 'PICTURE'
<p-type> c '.' ; y ::= <integer> d <variable> e 'PICTURE'
<p-type> f <R-subsection> g
$(2 \le a^v \le 49 \quad , \quad 2 \le d^v \le 49)$
<location> ^{l}
$b^l = ((x^{frn}, a^v), b)$; $e^l = ((y^{frn}, d^v), e)$
<type> $b^t = c^v$; $e^t = f^v$
<length> $b^n = c^n$; $e^n = f^n$
<record environment> $x^r = (b^l, b^t)$; $y^r = (e^l, e^t) \cup g^r$

<p-type> x ::= <p-code> a ; y ::= <p-letter> b '('
<integer> c ')' ; z ::= <decimal> d
<value> $x^v = a^v$; $y^v = b^v$; z^v = numeric
<length> $x^n = (a^n, 0)$; $y^n = (c^v, 0)$, $z^n = d^n$

<p-letter> x ::= 'A' ; y ::= 'X' ; z ::= '9'
<value> x^v = alphabetic ; y^v = alpha or numeric ;
z^v = numeric

Page 7

<p-code> x ::= <A-dupe> a ; y ::= <x-dupe> b ; z ::= <9-dupe> c

<value> x^v = alphabetic ; y^v = alphabetic or numeric ; c^v = numeric

<length> $x^n = a^n$; $y^n = b^n$; $z^n = c^n$

<A-dupe> x ::= 'A' ; y ::= 'A' <A-dupe> a

<length> $x^n = 1$; $y^n = a^n + 1$

<X-dupe> x ::= 'X' ; y ::= 'X' <X-dupe> a

<length> $x^n = 1$; $y^n = a^n + 1$

<9-dupe> x ::= '9' ; y ::= '9' <9-dupe> a

<length> $x^n = 1$; $y^n = a^n + 1$

<decimal> x ::= <9-code> a 'V' <9-code> b

<length> $x^n = (a^n, b^n)$

<9-code> x ::= <9-dupe> a ; y ::= '9' '(' <integer> b ')'

<length> $x^n = a^n$; $y^n = b^v$

<Working Storage Definition> x ::= <dep items> a ; y ::= <group items> b ; z ::= <indep items> c <group items> d

<effect> $x^e(s) = a^e(s)$; $y^e(s) = b^e(s)$; $z^e(s) = d^e(c^e(s))$

<indep items>w ::= '77' <variable>a 'PICTURE' <p-type>b '.' ; x ::= '77' <variable>c 'PICTURE' <p-type>d 'Value' <figurative constant>e ; y ::= '77' <variable>f 'PICTURE' <p-type>g '.' <indep items>h ; z ::= '77' <variable>i <p-type>j 'Value' <figurative constant>k <indep items>l

<effect> $w^e(s) = s$; $x^e(s) = s'$ where $s'(c) = e^v$ and $s'(p) = s(p)$ for $p \neq c$; $y^e(s) = h^e(s)$; $z^e(s) = s'(l^e(s))$ $s'(i) = k^v$ and $s'(p) = s(p)$ for $p \neq i$

<type> $a^t = b^v$; $c^t = d^v$; $f^t = g^v$; $i^t = j^v$

<length> $a^n = b^n$; $c^n = d^n$; $f^n = g^n$; $i^n = j^n$

<location> ^{l}

$a^l = ((\phi, 77), a)$; $c^l = ((\phi, 77), c)$; $f^l = ((\phi, 77), f)$; $i^l = ((\phi, 77), i)$

<group items> x ::= '01' <variable>a '.' <G-section>b

<effect> $x^e(s) = b^e(s)$

<file name> ^{fn}

$b^{fn} = a$

<G-section>w ::= <integer>a <variable>b 'PICTURE' <p-type>c '.' ; x ::= <integer>d <variable>e 'PICTURE' <p-type>f 'value' <figurative constant>g ; y ::= <integer>h <variable>i 'PICTURE' <p-type>j '.' <G-section>k ; z ::= <integer>l <variable>m 'PICTURE' <p-type>n 'value' <figurative constant>p <G-section>q

Page 9

$(2 \le d^v \le 49,\ 2 \le d^v \le 49,\ 2 \le h^v \le 49,\ 2 \le l^v \le 49)$

< effect > $w^e(s) = S$; $x^e(s) = S'$ where $S'(e) = g^v$ and $S'(r) = S(r)$ for $r \neq e$; $y^e(s) = k^e(s)$;
$z^e(q^e(s)) = S'$ where $S'(m) = p^v$ and $S'(r) = S(r)$ for $r \neq m$

< type > $b^t = c^v$; $e^t = f^v$; $i^t = j^v$; $m^t = n^v$

< length > $b^n = c^n$; $e^n = f^n$; $i^n = j^n$; $m^n = n^n$

< location > ^{l}

$b^l = ((w^{fn}, a^v), b)$; $e^l = ((x^{fn}, d^v, e)$;
$i^l = ((y^{fn}, h^v), i)$; $n^l = ((z^{fn}, l^v, m)$

< Open statement > x ::= 'OPEN' 'INPUT' < variable > a ;
y ::= 'OPEN' 'OUTPUT' < variable > b ;
z ::= 'OPEN' 'INPUT' < variables > c 'OUTPUT'
< variables > d
< position > ^{p}
$a^p = o$; $b^p = o$; $c^p = o$; $d^p = o$
< file type > a^f = input ; b^f = output ;
c^f = input , d^f = output

< Read statement > x ::= 'READ' < variable > a 'AT END'
< unlabeled P statement > b
< effect > $x^e(s) = S'$ where if $S(a) = (x_1, x_2, \ldots, x_n)$ then $S'(a) = (x_2, \ldots, x_n)$ and $S'(a^\wedge) = x_1$ and $S'(z) = S(z)$ for $z \neq a$ and $z \neq a^\wedge$
< record > ^{r}
< exit > x^x = if not end of file on a then x^x else b^x

<GO TO Statement> x ::= 'GO TO' <variable> a
<statement index> ^{i}
<label function> ^{a}
<effect> $x^{e}(s) = s$
<exit index> $x^{x}(s) = a^{a}$

<Add Statement> x ::= <Add part> a 'TO' <variable> b ;
y ::= <Add part> e <variable> c 'GIVING' <variable> d
<effect> $x^{e}(s) = s'$ where $s'(b) = b^{v} + a^{v}$ and
$s'(z) = s(z)$ for $z \neq b$; $y^{e}(s) = s'$ where $s'(d) = e^{v} + c^{v}$
and $s'(z) = s(z)$ for $z \neq d$.
<exit index> $x^{x}(s) = b^{x}(s)$; $y^{x}(s) = d^{x}(s)$

<Add Part> x ::= 'ADD' <data item> b ; y ::=
<add part> c <data item> d
<value> $x^{v}(s) = b^{v}(s)$; $y^{v}(s) = c^{v}(s) + d^{v}(s)$

<Subtract statement> x ::= <Subtract part> a 'FROM'
<variable> b ; y ::= <subtract part> c 'FROM'
<data item> d 'GIVING' <variable> e
<effect> $x^{e}(s) = s'$ where $s'(b) = b^{v} - a^{v}$, $s'(z) = s(z)$
for any $z \neq b$; $y^{e}(s) = s'$ where $s'(e) = d^{v} - c^{v}$,
$s'(z) = s(z)$ for $z \neq e$.
<exit index> $x^{x}(s) = b^{x}(s)$, $y^{x}(s) = e^{x}(s)$

<Subtract Part> x ::= 'SUBTRACT' <data item> a ; y ::=
<subtract part> b <data item> c
<value> $x^{v}(s) = b^{v}(s)$; $y^{v}(s) = b^{v}(s) + c^{v}(s)$

Page 11

<Multiply Statement>x ::= 'Multiply' <data item>a 'BY' <variable>b ; y ::= 'Multiply' <data item>c 'BY' <data item>d 'GIVING' <variable>e

<effect> $x^e(s) = s'$ where $s'(b) = a^V \times b^V$, $s'(z) = s(z)$ for $z \neq b$; $y^e(s) = s'$ where $s'(e) = c^V \times d^V$, $s'(z) = s(z)$ for $z \neq e$

<exit index> $x^X(s) = b^X(s)$; $y^X(s) = e^X(s)$

<Divide Statement>x ::= 'Divide' <data item>a 'INTO' <variable>b ; y ::= 'Divide' <data item>c 'INTO' <data item>d 'GIVING' <variable>e

<effect> $x^e(s) = s'$ where $s'(b) = b^V / a^V$, $s'(z) = s(z)$ for $z \neq b$; $y^e(s) = s'$ where $s'(e) = d^V / c^V$, $s'(z) = s(z)$ for $z \neq e$.

<exit index> $x^X(s) = b^X(s)$; $y^X(s) = e^X(s)$

<If Statement>x ::= 'IF' <predicate>V <imperative block>a <else> <imperative block>b

<effect> $x^e(s) =$ if $v^V(s)$ then $a^e(s)$ else $b^e(s)$

<exit index> $x^X(s) =$ if $v^V(s)$ then $a^X(s)$ else $b^X(s)$

<else>x ::= 'else' ; y ::= 'otherwise'

<Imperative Block>x ::= <imperative statement>a ; y ::= <imperative statement>b <imperative block>c

<effect> $x^e(s) = a^e(s)$; $y^e(s) = c^e(b^e(s))$

<exit index> $x^X(s) = a^V(s)$; $y^X(s) = c^X(s)$

<Predicate>w ::= <AND PREDICATE>a ; x ::= <OR PREDICATE>b ; y ::= <simple predicate>c ; z ::= <simple predicate>d

<value> $w^V = a^V$; $x^V = b^V$; $y^V = c^V$; $z^V = d^V$

<And predicate>x ::= <simple predicate>a 'AND' <simple predicate>b ; y ::= <simple predicate>c 'AND' <AND predicate>d

<value> $x^V = a^V \wedge b^V$; $y^V = c^V \wedge d^V$

<OR predicate>x ::= <simple predicate>a 'OR' <simple predicate>b ; y ::= <simple predicate>c 'OR' <OR predicate>d

<value> $x^V = a^V \vee b^V$; $y^V = c^V \vee d^V$

<simple predicate 1>u ::= <variable>a 'IS POSITIVE' ; v ::= <variable>b 'IS NEGATIVE' ; w ::= <variable>c 'IS ZERO' ; x ::= <variable>d 'IS NOT POSITIVE' ; y ::= <variable>e 'IS NOT NEGATIVE' ; z ::= <variable>f 'IS NOT ZERO'

<value> u^V = if $a^V > 0$ then true else false ;
v^V = if $b^V < 0$ then true else false ;
w^V = if $c^V = 0$ then true else false ;
x^V = if $d^V > 0$ then false else true ;
y^V = if $e^V < 0$ then false else true ;
z^V = if $f^V = 0$ then false else true

<simple predicate 2>w ::= <variable>a 'IS' <type pred>b ; x ::= <variable>c 'IS NOT' <type

pred> d ; x ::= <variable>e 'IS' <comparison-
son predicate>f <variable>g ; z ::=
<variable> h 'IS NOT' <comparison predicate> j
<variable> k
<value> w^v = if $a^t = b^v$ then true else false;
x^v = if $a^t = b^v$ then false else true;
y^v = if apply(f^v (e^v, g^v)) is true then
true else false; z^v = if apply (j^v (b^v, k^v))
is true then false else true

<simple predicate> x ::= <simple predicate 1> a ; y ::=
<simple predicate 2> b
<value> $x^v = a^v$; $y^v = b^v$

<type pred> x ::= 'NUMERIC' ; y ::= 'alphabetic'
<value> x^v = 'NUMERIC' ; y^v = 'alphabetic'

<comparison predicate> v ::= '>' ; w ::= 'GREATER
THAN' ; x ::= '<' ; y ::= 'LESS THAN' ; z ::=
'EQUAL TO'
<value> $v^v = >$; $w^v = >$; $x^v = <$; $y^v = <$;
$z^v = =$

<Arithmetic Go to statement> x ::= 'GO TO' <paragraph
list>a 'DEPENDING ON' <variable> b
<label function>^{a}
<exit index> x^x = if $1 \leq b^v \leq a^n$ then $x^a(a)$
else error

< paragraph list > x ::= < variable > a ; y ::=
< paragraph list > b ',' < variable > c
< length > $x^n = 1$; $y^n = b^n + 1$
< function of length > $x^{f(1)} = a$; $y^{f(b^n+1)} = c$

< Stop Statement > x ::= 'STOP' < literal >
< exit > x^x = undefined

< Alter Statement > x ::= 'ALTER' < variable > a 'TO
PROCEED TO' < variable > b
< exit index > ^{x}
$a^x = b$

< Perform Statement > x ::= < simple perform > a ;
y ::= < simple perform > b < cond perform > c
< effect > $x^{e(s)} = a^{e(s)}$
< iterated effect > ^{f}
< exit > y^x = if c^v then b else $y^n + 1$

< simple perform > x ::= 'PERFORM' < variable > a ;
y ::= 'PERFORM' < variable > b 'THRU' < variable > c
< exit > ^{x}
$a^x = x^x$; $c^x = y^x$
< effect > $x^{e(s)} = a^{e(s)}$
< iterated effect > ^{f}

< Cond perform > x ::= 'VARYING' < variable > a 'FROM'
< variable > b 'BY' < variable > c 'UNTIL' < predicate > d

; y ::= 'UNTIL' <predicate>^{e}
<conditional variable>^{e}
$a^{c} \cdot x^{c}$; $c^{c} = y^{c}$
<iterative effect>^{f}
$a^{f} = x^{r}$; $c^{f} = y^{c}$
<iterative exit index>^{n}:
$a^{n} = x^{n}$; $c^{n} = y^{n}$
<effect> $x^{e}(s)$ = if d^{v} then true else false ;
$y^{e}(s)$ = if e^{v} then true else false