SYSTEM 4/50 - 70 ALGOL

REPORT 2.

THE INTERMEDIATE CODE

4th May 1966.

L.W.Moore

INTRODUCTION

The design of the Intermediate Code generated by Phase 1Ø of the S4/50 Algol Compiler is based on that of the Object Program produced by the Whetstone KDF9 Algol Translator (as defined in 'Algol 60 Implementation' by B. Randell and L.J. Russell, Academic Press, 1964). There are however differences consequent upon (i) the three-pass structure of the S4/50 Algol Compiler, which is designed to generate machine code as the end product, and (ii) the altered method of handling the run-time stack, now based on the ideas of the ALCOR-ILLINOIS 7090 Compiler.

This document has a scope restricted to a definition of those parts of the S4/50 Algol Intermediate Code which differ from the corresponding parts of the Whetstone KDF9 Algol Object Program. It is a reference document only.

Further documents will deal with detailed aspects of the various Phases of the Compiler and of the generated object program.

CONTENTS

Section		Page
1.	LABELS	1.1
1A.	Note on Use of Labels and Switches	1.2
2.	BLOCKS	2.1
3.	PROCEDURES	3.1
3A.	Procedure Structure	3.1
3B.	Calls of Function Designators, and Procedure Statements	3.2
3C.	Parameter List Operations	3.3
3D.	Actual Operations	3.4
3E.	Use of Formal Parameters	3.9
4.	SWITCHES	4.1
4B.	Switch Elements and Array Elements as Actual Parameters	4.2
5.	CONDITIONAL STATEMENTS	5.1
6.	FOR STATEMENTS	6.1
6B.	Structure of for statement with three Elements	6.2
6C.	While element	6.3
6D.	Step until element	6.4
6E.	Example of a for statement	6.5
7.	OWN VARIABLES	7.1
8.	ARRAYS	8.1
9.	NOTE ON SEGMENTS AND MODULES	9.1
10.	DICTIONARY OF INTERMEDIATE CODE OPERATIONS	10.1
11.	BIT PATTERNS FOR THE INTERMEDIATE CODE OPERATIONS	11.1
12.	THE INTERMEDIATE CODE OPERATION FAIL	12.1

1, LABELS

Mnemonic labels appearing in the Position Identifier Table have the following significance:-

Gg --- internally generated labels
Ll --- user's own labels
Pp --- procedure labels

These labels are entered in the P.I.T. when and only when they occur as declarations (left-hand labels).

Throughout this document these labels, in both their left - and right - hand incarnations, have been written explicitly into the intermediate code for clarity of exposition.

In fact, left-hand labels have no existence in the intermediate code. They exist only as entries in the P.I.T.

Right-hand labels in the intermediate code are pointers to the appropriate entry in the P.I.T., and occupy two bytes. Denoting this two-byte pointer by p the label operations, with their parameters, are:-

```
TL (k,p),n          (Take label)
UJ (p)
 etc.               (Unconditional Jump)
```

Parameters such as (k,p) - written in the examples explicitly as e.g. (k,G5) - denote that k, p are packed into 2 bytes with 4 bits for k, 12 bits for p. This implies a maximum limit of 4096 labels in any module.

1 A) Note on Use of Labels and Switches

The translation of a designational expression into intermediate code will not explicitly involve use of the operations GTA, GTFA. Instead, the following convention is used:-

a) Within an actual parameter expression or a switch declaration:

The label operations TL, TFLN, TFLV, SINDR, and TFS have the run-time effect of stacking the resulting label

FFSk	
N	Address

b) Elsewhere:

i) The operation TL does not stack its label at run-time, but uses registers to effect an immediate transfer of control, using an implied GTA.

ii) The operations TFLN, TFLV, SINDR, and TFS, at run-time, all operate on the previously-stacked label, effecting an immediate transfer of control using an implied GTFA.

The operations GTA, GTFA therefore do not appear in the intermediate code.

This convention leads to some simplification of the intermediate code, as illustrated in the following example.

Example :-

'IF' A 'THEN' L1 'ELSE' S [2]

(A boolean, L1 label, S switch)

a) Within actual parameter expression, or switch

```
      BEX
      TBR (k,s)
      IFJ (G1)
      TL (k,L1),n
      UJ (G2)
G1:   SAPP
      TIC (2)
      SINDR (k,S)
G2:
```

b) Elsewhere

```
      BEX
      TBR (k,s)
      IFJ (G1)
      TL (k,L1),n
G1:   SAPP
      TIC (2)
      SINDR (k,S)
```

Case (a) is the normal Whetstone form of the translation, case (b) is the abbreviated form made possible by the convention described above.

2. BLOCKS

Within a hierarchy, block structure is represented in the Intermediate Code (I/C) by:-

BE (n) (BE ≡ Block Entry)

BEND (n) (BEND ≡ Block End)

n = block nesting number for the current hierarchy, starting from one at the start of the procedure body. For a labelled procedure body, n for the label is zero.

BE The action of BE at run time is to plant the current value of WP (working pointer) into word n of the Block Index List.

BEND The run-time action of BEND is to replace WP by the contents of word (n-1) of the Block Index List.

3. PROCEDURES

3A) Procedure Structure

```
     UJ (Gg)                                  )
Pp:  PE (k,L), m,N                            )
          )                                   )
          ) ---- parameter list operations    )
          )                                   )
          )                                   )      Declaration.
          )                                   )
          ) ---- procedure body               )
          )                                   )
          )                                   )
     RETURN                                   )
Gg:                                           )
```

```
     APP (k,π)                                )
          )                                   )
          )                                   )
          ) ---- actual operations            )      Call
          )                                   )
          )                                   )
                                              )
Gg': CF
```

where k = hierarchy number,
L = 1st order working storage for whole hierarchy (including link data and parameter space).
m = No. of words of parameter space,
N = maximum block depth within this hierarchy
π = index number of the procedure (th entry in list (a) of the level parameters). π = 1-16 for formal procedures.

N.B. The label on a CF operation is mandatory for diagnotic purposes, even if it is not used on a right hand side. Similarly, the label on a PE operation is mandatory.

3B) Calls of Function Designators, and Procedure Statements

The call operations for the various kinds of procedure are:-

CF	- Call function
CTF	- Call type function
CFF (k,s)	- Call formal function
CTFF (k,s)	- Call type formal function
CFZ (k,π)	- Call function zero
CTFZ (k,π)	- Call type function zero
CFFZ (k,s),π	- Call formal function zero
CTFFZ (k,s),π	Call type formal function zero.

The operations CF, CTF require no parameters, since they are always preceded by an APP operation whose parameter (π) gives the necessary information. In the case of calls of procedures which have no parameters (CFZ, etc) this information (π) has to appear as a parameter to the call operation itself. π is a pointer to the πth entry in level parameter table (a), which contains among other things a pointer to the entry for the procedure in the Position Identifier Table.

A function call which invokes a procedure as a function designator uses one of the operations

CTF, CTFZ(k,π), CTFF(k,s), CTFFZ(k,s),π.

The corresponding call invoking a procedure as a procedure statement uses one of the operations

CF, CFZ(k,π), CFF(k,s), CFFZ(k,s),π.

Use of a CF type operation (procedure statement) will imply a following REJECT operation. The effect of this implied REJECT is to delete the unused or superfluous result accumulator on exit from the called procedure

3C) Parameter List Operations

The only operations necessary are:-

CRFA	-	Copy real formal array	)
CIFA	-	Copy integer formal array	) Arrays by value
CBFA	-	Copy boolean formal array	)
CA	-	Check array	- All arrays by name
CP	-	Check parameter	- All other quantities by name or by value

3D) Actual Operations

At a procedure call the generated actual operations are:-

PSR (Gg)	Parameter subroutine		
PRA (k,s),-	Par. real array		
PIA (k,s),-	Par. integer array		
PBA (k,s),-	Par. boolean array		
PSW (k,p),-	Par. switch	PFSW (k,s),-	Par. formal switch
PPRπ,-,-	Par. procedure	PFPR (k,s),π	Par. formal procedure
PIFπ,-,-	Par. integer function	PFFI (k,s),π	Par. formal fn.integer
PRFπ,-,-	Par. real function	PFFR (k,s),π	Par. formal fn.real
PBFπ,-,-	Par. boolean function	PFFB (k,s),π	Par. formal fn.boolean
PST (<string>)	Par. string	PFST (k,s),-	Par. formal string
PIC (const)	Par. integer constant		
PRC (const)	Par. real constant		
PBC (const)	Par. boolean constant		
PI (k,s),-	Par. integer	PFIR (k,s),-	Par. formal integer
PR (k,s),-	Par. real	PFRR (k,s),-	Par. formal real
PB (k,s),-	Par. boolean	PFBR (k,s),-	Par. formal boolean
PL (k,p),n	Par. label	PFLN (k,s),-	Par. formal label (name)
		PFLV (k,s),-	Par. formal label (value)

'-' indicates a 1 byte space.

The label parameter to PSR is the label of the next actual operation or of the CF - type operation, whichever is appropriate. The subroutine heralded by PSR is always terminated by an EIS (End Implicit Subroutine) operation. PSR is used for any actual parameter whose evaluation requires more than one intermediate code operation (expressions, array elements, designational expressions). The body of an implicit subroutine contains ordinary I/C operations. (TRA, TFAB, CFFZ etc).

Actual Operations - General Structure of Procedure Call

	APP (k,π)	'Actual parameters' marker π = procedure no. (0-15 if formal procedure) APP absent if no parameters.
	PSR (G1)	Implicit subroutine
	EIS	
G1:	PRFπ,-,-	
	PBFπ,-,-	
	PSR (G2)	Implicit subroutine
	EIS	
G2:		
	PFIR (k,s),-	
G3:	CF	Procedure call.

Actual Operations - Example of a Procedure Call

```
P (A, 4, 'TRUE', X + Y, C, 'IF' I = ∅ 'THEN' R1 'ELSE' R2,
      S, Q, <PING>, F, B[1,4], T[2]);
```

The following types are assumed for the variables occurring in these actual parameters:-

A	real	
X	real	
Y	real	formal
C	real array	
I	integer	
R1	label	
R2	label	
S	switch	
Q	integer procedure	
F	procedure	formal
B	integer array	
T	switch	

```
        APP (k,π)                                   actual parameters marker
        PFR (k,s),-                                 A
        PIC (4)                                     4
        IMC (true)                                  true
        PSR (G1)                    )
        TRR (k,s)                   )
        TFR (k,s)                   )               X + Y
        +                           )
        EIS                         )

G1:     PRA (k,s),-                                 C
        PSR (G2)                    )
        DEX                         )
        TLR (k,s)                   )
        TIC Ø                       )               R1 or R2
        =                           )
        IFJ (G3)                    )
        TL (k,R1), n                )
        UJ (G4)                     )
G3:     TL (k,R2), n                )
G4:     EIS                         )
G2:     PSW (k,p),-                                 S
        PIF π,-,-                                   Q
        PST (<PING>)                                <PING>
        PFPR (k,s),π                                F
        PSR (G5)                    )
        TIAA (k,s)                  )
        TIC1                        )               B[1,4]
        TIC (4)                     )
        IMDR                        )
        EIS                         )
```

```
G5:     PSR (G6)            )
        SAPP                )
        TIC (2)             )       T [2]
        SINDR (k,T)         )
        EIS                 )

G6:     CF                          Call procedure P
```

3E) Use of Formal Parameters

a) By Value

The straightforward operations TIR, TRA, etc are used, the address parameters to which point to the appropriate formal accumulator. This is because parameters by value are regarded as local variables to the procedure.

b) By Name

TFAR (k,s) TFAI (k,s)	Addresses of arithmetic formal variables
TFR (k,s) TFI (k,s)	Values of arithmetic formal variables
TFAB (k,s)	Address of formal boolean
TFB (k,s)	Value of formal boolean
TFS (k,s)	Formal switch
CFF (k,s)	Formal procedure
CFFZ (k,s), π	Formal procedure zero
CTFF (k,s)	Type formal procedure
CTFFZ (k,s), π	Type formal procedure zero

c) Labels

TFLN (k,s)	By name.
TFLV (k,s)	By value

d) Arrays

Arrays used as formal parameters (by name or by value) require no special operations, since the array parameter is the array word.

4. SWITCHES

4A) This implementation depends on the fact that the complement of the switch index is handed over, from the call to the declaration of the switch, in a register.

The overall structure of a switch is as follows:-

```
        UJ (G1)
G2:     DSI (G3)        - -  Decrement switch index
        EIS             - -  Exits to the point of call
                             with a label result
G3:     DSI (G4)
        EIS
G4:     DSI (G5)
        EIS
G5:     ESL             - -  Failure, switch index
                             out of range
G1:
```

Switch Declaration

```
        SAPP            - -  Switch Approaching
        SINDR (k,G2)    - -  Switch index result
```

Switch Call

N.B. The first DSI operation at a switch declaration must be labelled.

4E) Switch elements and Array elements as Actual Parameters

If an actual parameter in a procedure call is written as e.g.

S [2]

it may not be known whether S is a switch or an array until later in the translation. It is therefore necessary to allow space for either interpretation in the I/C.

The alternative final translations are:-

switch element		array element
SAPP)		
DUMMY)	corresponds to	TRA (k,s)
DUMMY)		
SINDR (k,G2)	corresponds to	(INDR
		(DUMMY
		(DUMMY
TFS (k,s)	corresponds to	(INDR
		(DUMMY
		(DUMMY

5. CONDITIONAL STATEMENTS

The implementation of conditional statements in I/C is illustrated by the following examples:-

a) 'IF' A 'THEN' B 'ELSE' C

```
        BEX                         - - -    Boolean expression marker
                                    - - -    A
        IFJ (G1)

                                    - - -    B

        UJ (G2)
G1:
                                    - - -    C

G2:
```

b) 'IF' 'IF' A 'THEN' B 'ELSE' C 'THEN' D 'ELSE' E

```
        BEX
        BEX
                                    - - -    A

        IFJ (G1)

                                    - - -    B

        UJ (G2)
G1:
                                    - - -    C

G2:     IFJ (G3)

                                    - - -    D

        UJ (G4)
G3
                                    - - -    E

G4
```

6. FOR STATEMENTS

6A) General:

A for statement is converted into a normal block, which opens with the I/C operation

FBE (n)

and closes with

BEND (n)

The first order working storage of the for block is given by

I_{for} = 10 + first order working storage required by the controlled statement.

The value 10 arises from the need for six parameters which control the running of the various for list elements. These parameters are:-

address (k,n) :	FI	(integer)	:	the for index
" (k,n+1) :	TI	(integer)	:	the text index (initially TI = FI)
" (k,n+2) :	S1		)	various parameters needed
" (k,n+4) :	S2		)	for the implementation of step
" (k,n+6) :	S3		)	until elements (g.v.)
" (k,n+8) :	S4	(integer)	)	
" (k,n+9) :	spare		:	for alignment

The two I/C operations DFI (Gg) (Decrement For Index) and AST (Gg) (Avoid Step) are introduced, the expansions for which in I/C would be:

DFI (Gg)

{ TIA (k,n+1)
{ TIR (k,n+1)
{ TIC1
{ −
{ STA
{ TICØ
{ =
{ IFJ (Gg)

AST(Gg)

{ TIR (k,n + 8)
{ TICØ
{ =
{ IFJ (Gg)

GB) Structure of for statement with 3 elements

```
     BEGD (n)
     TI: = S4: = Ø

G1:  S2: = address of controlled
           variable v                    - - - -  (TS = Top of Stack)
     S3: = value of controlled
           variable v
     FI: = FI + 1
     TI: = FI
     DFI (G3)                            - - - -  Decrement TI and jump if ≠ Ø
                    )
                    )                    - - - -  First for list element
                    )                             (resulting value stored in
                    )                             pre-stacked address of v)
     ST             )
     UJ (G2)                             - - - -  jump to controlled statement

G3:  DFI (G4)
                    )
                    )
                    )                    - - - -  Second for list element
                    )
     ST

     UJ (G2)

G4:  DFI (G5)
                    )
                    )
                    )                    - - - -  Third for list element
                    )
     ST             )

G2:                 )
                    )
                    )
                    )
                    )                    - - - -  Controlled statement
                    )
                    )
                    )
                    )

     UJ (G1)                             - - - -  Return to G1 for next
                                                  element, or to continue
                                                  current element.
G5:  BEND (n)
```

6C) WHILE ELEMENT

e.g. 'FOR' V: = A 'WHILE' B 'DO' C ;

```
     FBE (n)
     FI: = S4: = Ø
G1:  TS: = address of V
     S3: = value of V
     FI: = FI + 1
     TI: = FI
     DFI (G3)
                    )
                    )                    --- Evaluate value of A
                    )
                    )
                    )
     ST                                  --- Store into V
     BEX            )
                    )
                    )
                    )                    --- Evaluate B
                    )
                    )
                    )
     IFJ (G1)                            --- Element exhausted
     FI: = FI - 1                        --- To prevent passage to next element
                                             until current element is exhausted
     UJ (G2) *                           --- To controlled statement
G2:                 )
                    )
                    )                    --- Controlled statement
                    )
                    )
                    )
     UJ (G1)
G3:  BEND (n)
```

* This instruction is omitted if the current element is the last (or only) element.

6D) STEP UNTIL ELEMENT

e.g. A step B until C

```
G2*:    DFI (G3)                    - - - -  To next DFI or BEND
        AST (G4)                    - - - -  Jump if S4 ≠ 0
        S1: = A
        TS: = S1
        S4: = 1
        S2: = B
        UJ (G5)                     - - - -  Jump around step

G4:     S2: = B
        S1: = S3 + S2
        TS: = S1

G5:     ST                          - - - -  V: = S1 = V + S2 or A
        BEX
        sign (S2) x (S1-C)
        TIGØ
        >
        IFJ (G6)
        S4: = Ø                     - - - -  Initialise marker S4
        UJ (G1)                     - - - -  Element exhausted

G6      FI: = FI -1
        UJ (G7) **                  - - - -  To controlled statement

G3:     DFI (G8)        )
                        )
                        )           - - - -  Next element
                        )
                        )
```

* Label G2 present if there was a previous DFI.

** Instruction UJ (G7) omitted if this is the last (or only) element.

6E) Example of a for statement

```
'FOR' A : = 1 'STEP' 2 'UNTIL' 10, 1, A + 2 'WHILE' A 'LT' 10
                                    'DO'  I : = A;
```

This statement is assumed to occur in block 3 of hierarchy 2, and the stack addresses are assumed to be:-

A	(2;14)	(real)
I	(2;17)	(integer)
FI	(2,18)	(integer)
TI	(2;19)	(integer)
S1	(2,20)	(integer)
S2	(2;22)	(integer)
S3	(2;24)	(real)
S4	(2,26)	(integer)

The translation is:-

```
        FDE (4)
        TIA (2,18)
        TIA (2,26)
        TICØ
        STA
        ST                          FI: = S4: = Ø

G1:     TRA (2;14)                  TS: = address of controlled
        TRA (2,24)                        variable A
        TRR (2,14)
        ST                          S3: = value of controlled variable A
```

```
                    TIA (2,18)
                    TIA (2,19)
                    TIR (2,18)
                    TIC1
                    +
                    STA
FIRST ELEMENT       ST                      FI: = TI: =  FI + 1
D'STEP'E'UNTIL'F:
                    DFI (G2)                Decrement For Index (TI)
                    AST (G3)                Avoid Step (jump if S4 ≠ 0).
                    TIA (2,20)
                    TIC1
                    STA                     S1: =  D, TS: = S1 =  D
                    TIA (2,26)
                    TIC1
                    ST                      S4: = 1
                    TIA (2,22)
                    TIC (2)
                    ST                      S2: =  E
                    UJ (G4)                 Jump around step

G3:                 TIA (2,20)
                    TIA (2,22)
                    TIC (2)
                    STA                     S2: =  E, TS: =  E
                    TRR (2,24)
                    +
                    STA                     S1: = S3 + S2, TS: = S1

G4:                 ST
                    TIC1                    Controlled variable V: = S1 = V
                                            + S2 or  D
                    TIR (2,22)
                    TICØ
                    ≥
```

```
                IFJ (G5)
                NEG
G5:             TIR (2,20)          TS: = sign (S2)
                TIC (10)
                 -
                 X
                TICØ
                 >
                IFJ(G6)             sign (S2) x (S1-C)  0 ?
                TIA (3,10)
                TICØ
                ST                  S4: = 0
                UJ (G1)             element exhausted
G6:             TIA (2,18)
                TIR (2,18)
                TIC1
                 -
                ST                  FI: = FI - 1
                UJ (G7)             To controlled statement
SECOND ELEMENT:
G2:             DFI (G8)
                TIC1
                ST                  Controlled variable V: =1
                UJ (G7)             To controlled statement

THIRD ELEMENT:
D'WHILE'E'DO'F:

G8:             DFI (G9)
                TRR (2,14)
                TIC (2)
                +
                ST                  Controlled variable V:= D = A + 2
```

```
                BEX
                TIR (2,14)
                TIC (10)

                IFJ (G1)                Jump if element exhausted
                TIA (2,18)
                TIR (2,18)
                TIC1
                -
                ST                      FI: = FI -1
```

CONTROLLED STATEMENT:

```
G7:             TIA (2,17)
                TRR (2,14)
                ST                      I : = A
                UJ (G1)                 Return for next element

G9:             BEND (4)
```

7. Own Variables

Identified as hierarchy -1 (i.e. k for an own variable is 4 bits of all ones).

Own variables at run time reside at the end of the program's (or module's) constant space. Space for all own variables used in a program is reserved immediately on entry to the program.

8. ARRAY

Where ... Algol uses the operat

MRSF (k, ,a

MISF (k,s), ,s),a

MBSF (k,s), a MOBSF (k,s) a

for real, integer, and boolean arrays respectively, where (k,s) is the stack address of the appropriate array word.

INDA, INDR remain unchanged, and are parameterless.

Further, to assist the optimisation of accessing elements of 1 dimensional arrays, the operations

TRAA (k,s)	Take real array address
TIAA (k,s)	Take integer array address
TBAA (k,s)	Take boolean array address

are introduced. These take the place of the operations TRA, TIA, TBA for obtaining the address of the array word.

9. NOTE ON SEGMENTS AND MODULES

The standard function

FETCH (<IDENT>);

will bring the segment <IDENT> into the store using the Control Routine FETCH. It will not enter any of the segment's constituent modules.

The specification of a module (i.e. an independently compiled procedure) within the calling program will be e.g.

'PROCEDURE' P(A); 'REAL' A;

'ENTER' <IDENT> , <LANGUAGE>;

where <IDENT> is the identifier of the module to be entered, and <LANGUAGE> specifies the language in which the module was written.

The call of this module will be written in the usual way, as P(X);

The fetch operation for a segment must have been performed in the program dynamically before any call of a constituent module of the segment, unless it is already in store by virtue of having been linked with a prior segment or program.

Any module not written in Algol (or 4/50 assembly code) has linked with it an environment module.

The action of the routine ENTER is to enter the environment module for the body concerned, if this body is in a language other than Algol, and to hand across the address (provided by "linkage editor") of the body being called. The environment module arranges the information available (stack, etc.) in the form required by the body, plants a return address into a place recognised by the body, and transfers control to the body.

After the body has been obeyed its normal return mechanism will transfer control back to the environment module, which then resets the information available into the form required by the Algol base program, and finally performs an ordinary procedure return to the point of call of the body by working on the links in the Algol stack.

10. DICTIONARY OF INTERMEDIATE CODE OPERATIONS

Operation		Parameters
AOA	Avoid Own Array	p
APP	Actual Parameters	k,π
AST	Avoid Step	p
BE	Block Entry	n
BEND	Block End	n
BEX	Boolean Expression	-
CA	Check Array	-
CBFA	Copy Boolean Formal Array	-
CF	Call Function	-
CFF	Call Formal Function	(k,s)
CFFZ	Call Formal Function Zero	(k,s),π
CFZ	Call Function Zero	k,π
CIFA	Copy Integer Formal Array	-
CP	Check Parameter	-
CRFA	Copy Real Formal Array	-
CTF	Call Type Function	-
CTFF	Call Type Function Formal	(k,s)
CTFFZ	Call Type Function Formal Zero	(k,s),π
CTFZ	Call Type Function Zero	k,π
DFI	Decrement For Index	p
DSI	Decrement Switch Index	p
DUMMY	Dummy	-

Operation		Parameters
EIS	End Implicit Subroutine	-
ESL	End Switch List	-
FBE	For Block Entry	n
FINISH	Finish	-
IFJ	If False Jump	p
INDA	Index Address	-
INDR	Index Result	-
MBSF	Make Boolean Storage Function	(k,s),a
MISF	Make Integer Storage Function	(k,s),a
MOBSF	Make Own Boolean Storage Function	(k,s),a
MOISF	Make Own Integer Storage Function	(k,s),a
MORSF	Make Own Real Storage Function	(k,s),a
MRSF	Make Real Storage Function	(k,s),a
NEG	Negate	-
PB	Parameter Boolean	(k,s),-
PBA	Parameter Boolean Array	(k,s),-
PBC	Parameter Boolean Constant	constant
PBF	Parameter Boolean Function	π,-,-
PE	Procedure Entry	(k,L),m,N
PFBR	Parameter Formal Boolean Result	(k,s),-
PFFB	Parameter Formal Function Boolean	(k,s),π
PFFI	Parameter Formal Function Integer	(k,s),π
PFFR	Parameter Formal Function Real	(k,s),π

Operation		Parameters
PFIR	Parameter Formal Integer Result	(k,s),-
PFLN	Parameter Formal Label (Name)	(k,s),-
PFLV	Parameter Formal Label (Value)	(k,s),-
PFRR	Parameter Formal Real Result	(k,s),-
PFST	Parameter Formal String	(k,s),-
PFSW	Parameter Formal Switch	(k,s),-
PFPR	Parameter Formal Procedure	(k,s),π
PI	Parameter Integer	(k,s),-
PIA	Parameter Integer Array	(k,s),-
PIC	Parameter Integer Constant	constant
PIF	Parameter Integer Function	π,-,-
PL	Parameter Label	(k,p),n
PPR	Parameter Procedure	π,-,-
PR	Parameter Real	(k,s),-
PRA	Parameter Real Array	(k,s),-
PRC	Parameter Real Constant	constant
PRF	Parameter Real Function	π,-,-
PSR	Parameter Subroutine	p
PST	Parameter String	'string'
PSW	Parameter Switch	(k,p),-
RETURN	Return	-
SAPP	Switch Approaching	-
SINDR	Switch Index Result	(k,p)

Operation		Parameters
ST	Store	-
STA	Store Also	-
TBA	Take Boolean Address	(k,s)
TBAA	Take Boolean Array Address	(k,s)
TBCF	Take Boolean Constant False	-
TBCT	Take Boolean Constant True	-
TBR	Take Boolean Result	(k,s)
TFAB	Take Formal Address Boolean	(k,s)
TFAI	Take Formal Address Integer	(k,s)
TFAR	Take Formal Address Real	(k,s)
TFB	Take Formal Boolean	(k,s)
TFI	Take Formal Integer	(k,s)
TFLN	Take Formal Label (Name)	(k,s)
TFLV	Take Formal Label (Value)	(k,s)
TFR	Take Formal Real	(k,s)
TFS	Take Formal Switch	(k,s)
TIA	Take Integer Address	(k,s)
TIAA	Take Integer Array Address	(k,s)
TIC	Take Integer Constant	constant
TICØ	Take Integer Constant Zero	-
TIC1	Take Integer Constant One	-
TIR	Take Integer Result	(k,s)
TL	Take Label	(k,p),n

Operation		Parameters
TRA	Take Real Address	(k,s)
TRAA	Take Real Array Address	(k,s)
TRC	Take Real Constant	constant
TRR	Take Real Result	(k,s)
UJ	Unconditional Jump	p
+		
-		
*		
/		
'/'		
'**'		
'GT'		
'GE'		
=		
'NE'		
'LE'		
'LT'		
'NOT'		
'AND'		
'OR'		
'IMPL'		
'EQUIV'		
FAIL	Compilation Failure	r

PARAMETER NOTATION

p	p'th entry in Position Identifier Table ($1 \leq p$) (2 bytes)
(k,p)	k and p packed into 2 bytes with 4 bits for k and 12 bits for p, where k is the hierarchy number and p is the packed form of the p defined above ($0 \leq k$)
(k,s)	k and s packed into 2 bytes with 4 bits for k and 12 bits for s, where s is the number of words up the stack from the start of the link data for hierarchy k. (Stack Address).
(k,L)	k and L packed into 2 bytes with 4 bits for k and 12 bits for L, where L is the number of words of first order working storage (including parameter space and link data) for the hierarchy and its constituent blocks.
n	n'th block level in current (or destination) hierarchy. (1 byte) ($1 \leq n$)
π	π'th entry in list (a) of the Level Parameters (1 byte) ($1 \leq \pi$)
m	Number of words of parameter space (1 byte)
N	Total block nesting depth within current hierarchy. (1 byte) ($1 \leq N$)
constant	1 word for integer or boolean constants, 2 words for real constants.
'string'	Basic symbol representation (unpacked) of the string, including the opening and closing string quotes.
- (minus sign)	One-byte space
a	Number of arrays in an array segment (1 byte).
r	r'th failure for this module (1 byte). ($1 \leq r$)

11. BIT PATTERNS FOR THE INTERMEDIATE CODE OPERATIONS

CLASS	SUB-CLASS	TYPE	OPERATION	NUMERIC
000	000	00	FINISH	000
000	000	01	PE	001
000	000	10	RETURN	002
000	001	00	BE	010
000	001	01	BEND	011
000	001	10	FBE	012
000	010	00	DSI	020
000	010	01	ESL	021
000	011	10	DFI	032
000	011	11	AST	033
000	100	00	AOA	040
000	101	00	EIS	050
000	110	00	SAPP	060
000	110	01	APP	061
000	111	00	CP	070
000	111	01	CA	071
000	111	11	FAIL	073
001	000	00	RO	100
001	000	01	TRR	101
001	000	10	TIR	102
001	000	11	TBR	103
001	010	01	TFR	121
001	010	10	TFI	122
001	010	11	TFB	123
001	100	01	TRC	141
001	100	10	TIC	142
001	101	10	TICØ	152
001	101	11	TBCT	153
001	110	10	TIC1	162
001	110	11	TBCF	163
010	000	00	AO	200
010	000	01	TRA	201
010	000	10	TIA	202
010	000	11	TBA	203
010	010	01	TFRA	221
010	010	10	TFIA	222
010	010	11	TFBA	223
010	100	01	TRAA	241
010	100	10	TIAA	242
010	100	11	TBAA	243

CLASS	SUB-CLASS	TYPE	OPERATION	NUMERIC
011	000	01	MRSF	301
011	000	10	MISF	302
011	000	11	MBSF	303
011	001	01	MORSF	311
011	001	10	MOISF	312
011	001	11	MOBSF	313
011	010	01	CRFA	321
011	010	10	CIFA	322
011	010	11	CBFA	323
011	011	00	INDA	330
011	100	00	INDR	340
011	101	00	SINDR	350
011	110	00	TFS	360
100	000	00	FO or CF	400
100	000	01	CTF	401
100	010	00	CFZ	420
100	010	01	CTFZ	421
100	100	00	CFF	440
100	101	01	CTFF	441
100	110	00	CFFZ	460
100	110	01	CTFFZ	461
101	000	00	TL	500
101	010	00	TFLV	520
101	010	01	TFLN	521
101	100	01	UJ	541
101	100	10	IFJ	542
110	000	00	PO	600
110	000	01	PR	601
110	000	10	PI	602
110	000	11	PB	603
110	001	01	PRC	611
110	001	10	PIC	612
110	001	11	PBC	613
110	010	01	PFRR	621
110	010	10	PFIR	622
110	010	11	PFBR	623

CLASS	SUB-CLASS	TYPE	OPERATION	NUMERIC
110	011	01	PRA	631
110	011	10	PIA	632
110	011	11	PBA	633
110	100	00	PPR	640
110	100	01	PRF	641
110	100	10	PIF	642
110	100	11	PBF	643
110	101	00	PFPR	650
110	101	01	PFFR	651
110	101	10	PFFI	652
110	101	11	PFFB	653
110	110	00	PSR	660
110	110	01	PL	661
110	110	10	PFLN	662
110	110	11	PFLV	663
110	111	00	PST	670
110	111	01	PFST	671
110	111	10	PSW	672
110	111	11	PFSW	673
111	000	00	+	700
111	000	01	-	701
111	000	10	*	702
111	000	11	/	703
111	001	00	'/'	710
111	001	01	'**'	711
111	010	00	'GT'	720
111	010	01	'LT'	721
111	011	00	'GE'	730
111	011	01	'LE'	731
111	011	10	'NE'	732
111	011	11	=	733
111	100	00	'NOT'	740
111	100	01	'NEG'	741
111	101	00	'AND'	750
111	101	01	'OR'	751
111	101	10	'IMPL'	752
111	101	11	'EQUIV'	753
111	110	00	ST	760
111	110	01	STA	761
111	110	10	BEX	762
111	111	11	DUMMY	773

Reversing Fields, order of operations becomes:

Operation	Decimal	Operation	Decimal	Operation	Decimal	Operation	Decimal
FINISH	0	CP	56	PFFR	110	PFFI	174
RO	1	PST	62	'OR'	111	'IMPL'	175
AO	2	PE	64	APP	112	TIC1	177
FO or CF	4	TRR	65	CTFFZ	116	PFLN	182
TL	5	TRA	66	PL	118	BEX	183
PO	6	MRSF	67	STA	119	PSW	190
+	7	CTF	68	CA	120	TBR	193
BE	8	PR	70	PFST	126	TBA	194
'/'.	15	-	71	RETURN	128	MBSF	195
DSI	16	BEND	72	TIR	129	PB	198
CFZ	20	MORSF	75	TIA	130	/	199
TFLV	21	PRC	78	MISF	131	MOBSF	203
'GT'	23	'**'	79	PI	134	PBC	206
INDA	27	ESL	80	*	135	TFB	209
'GE'	31	TFR	81	FBE	136	TFBA	210
AOA	32	TFRA	82	MOISF	139	CBFA	211
INDR	35	CRFA	83	PIC	142	PFBR	214
CFF	36	CTFZ	84	TFI	145	AST	216
PPR	38	TFLN	85	TFIA	146	PBA	222
'NOT'	39	PFRR	86	CIFA	147	=	223
EIS	40	'LT'	87	PFIR	150	TBAA	226
SINDR	43	PRA	94	DFI	152	PBF	230
PFPR	46	'LE'	95	PIA	158	TBCT	233
'AND'	47	TRC	97	'NE'	159	PFFB	238
SAPP	48	TRAA	98	TIC	161	'EQUIV'	239
TFS	51	CTFF	100	TIAA	162	TBCF	241
CFFZ	52	UJ	101	IFJ	165	PFLV	246
PSR	54	PRF	102	PIF	166	PFSW	254
ST	55	'NEG'	103	TICØ	169	DUMMY	255
						total	117

A new operation FAIL has been added whose decimal representation is 248

12. The Intermediate Code Operation FAIL

Any syntactic failure detected by Phase 10 results in

a) a failure message to a listing device, and

b) the compilation of the intermediate code operation

FAIL (r)

where the current failure is the r'th to be detected.

The translation then continues (including the generation of intermediate code).

Phase 20 translates this operation into an entry into the run-time failure routine. The result is that at run-time the first of these entries to the failure routine to be dynamically encountered causes the listing of a failure message and the termination of the run. The failure message will include the identifier of the module in which the failure occurred.

The justification of this process is that the user in many cases will be able to obtain satisfactory results from his run before the error condition is encountered.

R. J. B.

S4-50/70 ALGOL

REPORT 2, THE INTERMEDIATE CODE

AMENDMENT I. 17th JUNE 1966

page 3.4

The parameters to the four operations PPR, PIF, PRF, and PBF have been changed to:-

PPR k, π, -

PIF k, π, -

PRF k, π, -

PBF k, π, -

These operations are also used on pages 3.5, 3.7, 10.2, 10.3.

L. W. Moore

L.W.Moore.