CONTROL DATA

CORPORATION

BULLETIN

APPLICATIONS DEVELOPMENT

3100/3200/3300/3500 ALGOL ABNORMAL OBJECT TIME TERMINATION DUMP

3100/3200/3300/3500 ALGOL
ABNORMAL OBJECT TIME
TERMINATION DUMP

CONTENTS

INTRODUCTION

This bulletin describes Object Time Abnormal Termination Dump in Control Data® 3100/3200/3300/3500 computer series ALGOL. It is assumed that the reader is acquainted with the general characteristics of 3100/3200/3300/3500 series computers and programming in the ALGOL language. The language is as defined in the Communications of the ACM, 1963, vol. 6, No. 1, pp 1-17 with some additional procedures and exceptions as stated in the Control Data Reference Manual.

OBJECT TIME ABNORMAL TERMINATION DUMP

Upon abnormal termination of an object program, a diagnostic (such as, ARITHMETIC OVERFLOW or FORMAT STRING ERROR) is printed on the standard output unit to indicate the nature of the error. The contents of all non-empty output format areas are output on their respective units. In particular, if there is a non-empty format area associated with LU 61 (standard output), its contents appear on that unit following the object time diagnostic. This information is followed by a structured dump.

STRUCTURED DUMP

The structured dump traces the execution path through the blocks in the block structure currently active when the error occurs. The information relevant to the ALGOL program at the time the error occurred (values, descriptions, and/or locations of variables), is selected from core storage for printing in this dump. The dump has the following format:

```
THIS ERROR OCCURRED AFTER        LINE xxxx
IN THE BLOCK ENTERED AT          LINE xxxx
(global information)
(environmental information)
THIS BLOCK WAS CALLED FROM       LINE xxxx
IN THE BLOCK ENTERED AT          LINE xxxx
(environmental information)
THIS BLOCK WAS CALLED FROM       LINE xxxx
IN THE BLOCK ENTERED AT          LINE xxxx
(environmental information)
     . . . . . . . . . . .
     . . . . . . . . . . .
```

The line number xxxx refers to the number assigned to each source image line during compilation and printed with the source program listing. If the block entered is a standard procedure, the word STAN appears instead of the line number.

GLOBAL AND ENVIRONMENTAL INFORMATION

Each line of global and environmental information consists of a 15-bit address field printed as 5 octal digits. This is immediately followed by 48 bits representing the contents of one stack entry, printed as 16 octal digits in fields of 2, 6, 2, and 6, as follows:

Address Field	Information Field			
xxxxx	xx	xxxxxx	xx	xxxxxx

GLOBAL INFORMATION

The global information applies to the running program as a whole, without regard to the currently active block structure. It has the following format:

THE GLOBAL VARIABLES ARE . .					
UA, VALUE	xxxxx	xx	xxxxxx	xx	xxxxxx
UV	xxxxx	xx	xxxxxx	xx	xxxxxx
LASTUSED	xxxxx	xx	xxxxxx	xx	xxxxxx

UA, UV, and LASTUSED are the names of variables internal to the ALGOL system.

UA

UA contains the address of the last accessed formal parameter, the address of the value of a typed procedure, or the address of the last referenced array element. The address field gives the contents of UA. The other 48 bits are the contents of the location referenced by this address.

UV

UV is used only to contain either the value of the last accessed formal parameter if this does not appear in the stack (such as, a formal expression) or the value of a typed procedure. Whenever UV is in use, UA contains the address of UV. The address field gives the address of UV. The other 48 bits are its contents.

LASTUSED

LASTUSED contains the address of the top stack element. The address field gives the address of the top stack element. The other 48 bits are the contents of the location referenced by this address.

ENVIRONMENTAL INFORMATION

The environmental information consists of descriptions or values of those formal and/or local variables belonging to the appropriate block level. Formal variables appear only if the particular block is a procedure. Simple local variables and simple formal parameters called by value are represented by their values; all other variables are represented by a description. The formats of these values and descriptions are given later.

FORMAL VARIABLES

Formal variables are dumped in the following structure:

1st line	Return information
2nd line	1st formal parameter
3rd line	2nd formal parameter
....	.....
....	.....
....	last formal parameter
....	1st constant used as actual parameter
....	2nd constant used as actual parameter
....	.....
....	.....

LOCAL VARIABLES

In addition to every declared variable, one stack entry exists for each artificial label generated for a for statement and one for each designational expression of a switch list; moreover, each bound-pair list, in an array declaration containing n bound pairs, generates n+1 stack entries. All of these entries appear in the stack in reverse order from their appearance in the source program, and they are dumped in this form. Any additional stack entries following the first declared (last printed) variables represent intermediate working locations generated by the compiler.

VALUES OF VARIABLES

Simple local variables and simple formal parameters called by value are represented in the stack as follows:

<u>Boolean</u>

A 48-bit entry in which bits 47-25 are always set to 0. Bit 24 is set to 1 for <u>true</u> and 0 for <u>false</u>. Bits 23-0 are irrelevant.

<u>Integer</u>

A 48-bit entry in a fixed-point, right-justified integer form.

<u>Real</u>

A 48-bit entry in standard floating-point form.

DESRIPTIONS OF VARIABLES

All descriptions of variables in the stack have the following general form:

$$\langle x \rangle_3 \ \langle t \rangle_3 \ \langle \text{address } 1 \rangle_{18} \ \langle i \rangle_1 \ \langle kk \rangle_5 \ \langle \text{address } 2 \rangle_{18}$$

x is three bits representing the transformation which must be applied to formal arithmetic variables.

t is three bits representing the type of a variable.

i is 1 bit used by the system in conjunction with kk as described below.

kk is 5 bits representing the kind of the variable.

The interpretation of the addresses address 1 and address 2 depends on the kind (kk) of description.

The transformation, x, is used only for the descriptions of formal parameters and can take the following values:

		Possible Use
0	No transformation	Formal and local
1	Fix	Formal only
2	Float	Formal only
3	Fix-then-float	Formal only
4-7	Not used	Not used

The type, t, can take the following values:

		Possible Use
0	No type	Formal and local
1	Boolean	Formal and local
2	Real	Formal and local
3	Integer	Formal and local
4	Real-integer	Formal only
5	Integer-real	Formal only
6	Real-integer-real	Formal only
7	Not used	Not used

The kind, kk, can take the following values:

		Possible Use
00	Not used	Not used
01	Not used	Not used
02	Switch	Formal and local
03	String	Formal and local
04	Label or designational expression	Formal and local
05	No-type procedure	Formal and local
06	Typed procedure	Formal and local
07	Array	Formal and local
10	Constant	Formal only
11	Expression	Formal only
12	Simple variable	Formal only
13	Subscripted variable	Formal only
14-37	Not used	Not used

The reader should be aware that a stack entry representing an arithmetic value may have a bit structure which makes it appear to be a description.

DESCRIPTIONS 5

The following detailed explanations of the descriptions are ordered according to the kind, kk, as described above, except for Return Information which does not have a kind and is described first.

TERMINOLOGY

All references to the stack in the object program are relative to the beginning of the stack area for a particular block. When a block is entered at execution time, the base address of the corresponding stack area is assigned. This absolute base address is the Stack Reference of the block. In the following descriptions, Stack Reference is used to define the environment of the particular element.

The term Segment Location means an address pointing to a position in the object program. In non-segmented execution, it is the 18-bit complement of an absolute address. In segmented execution, it is interpreted as a 9-bit segment number followed by a 9-bit segment relative address.

The term Stack Address means an absolute address pointing to a particular stack entry.

RETURN INFORMATION

$<\text{No. of formals}>_6 \; <\text{Stack Reference}>_{18} \; <\text{No. of constants+1}>_6$

$<\text{Segment Location}>_{18}$

02 SWITCH

$<0>_3 \; <0>_3 \; <\text{No. of switch elements}>_{18} \; <0>_1 \; <02>_5 \; <1>_3 \; <\text{Stack Address}>_{15}$

Stack Address points to the first element of the switch list.

In a switch declaration, the switch list (see kind 04, below) precedes the switch description as follows:

$<\text{Designational expression of the nth switch element}>_{48}$

$<\text{Designational expression of (n-1)th switch element}>_{48}$

. . . .

. . . .

STACK ADDRESS <Designational expression of 1st switch element>$_{48}$

< Switch description as above >$_{48}$

03 STRING

<Relative Address>$_{12}$ <No. of string char.>$_{12}$ <i>$_{1}$ <03>$_{5}$ <Segment Location>$_{18}$

The above description is a copy of the description saved as an own variable in the stack. The Relative Address is the address relative to the stack reference of Block 0 of this own stack entry.

The string itself is stored in-line in the object program. The Segment Location points to the first word of the string.

After syntax checking on the string has been performed the first time, the i bit is set to 1 in the own stack entry, to prevent further syntax checking on the same string.

04 LABEL OR DESIGNATIONAL EXPRESSION

<0>$_{3}$ <0>$_{3}$ <Stack Reference>$_{18}$ <i>$_{1}$ <04>$_{5}$ <Segment Location>$_{18}$

If a designational expression is not a label, the Segment Location points to the code which evaluates the expression and jumps to the resulting label.

For each for statement, the compiler generates an artificial label which has the same description as above. This label is used to return from the end of the for statement to the control at the beginning. Whenever the for statement is not in execution, the Segment Location of this label is set to point to a special system entry 000011, in order to detect abnormal use of the statement. The i bit is set to 1 in this case.

In addition, the i bit is preset to 1 before entry to each step-until element, and set to 0 after this element has been entered.

05 NO-TYPE PROCEDURE

<0>$_{3}$ <0>$_{3}$ <Stack Reference>$_{18}$ <1>$_{1}$ <05>$_{5}$ <Segment Location>$_{18}$

06 TYPED PROCEDURE

<x>$_{3}$ <t>$_{3}$ <Stack Reference>$_{18}$ <1>$_{1}$ <06>$_{5}$ <Segment Location>$_{18}$

07 ARRAY

<x>$_{3}$ <t>$_{3}$ <Stack Address 1>$_{18}$ <0>$_{1}$ <07>$_{5}$ <1>$_{3}$ <Stack Address 2>$_{15}$

Stack Address 1 is the base address of the array elements in the stack.

The elements of an array are assigned above the last working location of the particular block, but do not appear in the dump.

<u>own</u> arrays are handled in the same way, except that their elements are assigned among the <u>own</u> variables in block 0.

The elements of an array called by value are copied (and transformed, if necessary) to a position above the working locations of the block of the procedure.

Stack Address 2 is the base address of the dope vector which is used to calculate the addresses of the array elements (see below).

In an array declaration, the dope vector of the corresponding bound-pair list precedes the descriptions for all array identifiers of an array segment.

The dope vector for the array declaration

$$\underline{\text{array}}\ A\left[\ell_1 : u_1,\quad \ell_2 : u_2, \ldots \quad \ell_n : u_n\right] \quad \text{is:}$$

$<C_n>_{24}$	$<\text{Not used}>_{24}$
$<C_{n-1}>_{24}$	$<\text{Not used}>_{24}$
.....	....
.....	....
$<C_2>_{24}$	$<\text{Not used}>_{24}$
$<\text{Length of array}>_{24}$	$<\text{Not used}>_{24}$
$<\text{Lower bound effect}>_{24}$	$<\text{n=No. of dimensions}>_{24}$

where $C_i = u_i - \ell_i + 1$

Length of array $= C_1 * C_2 * C_3 \ldots\ldots C_n$

Lower bound effect $= (((\ldots(\ell_1 * C_2 + \ell_2) * C_3 + \ell_3) * \ldots\ldots) * C_n + \ell_n$

The address of any element is referenced by the base address of the array plus

$$(((\ldots(i_1 * C_2 + i_2) * C_3 + i_3) * \ldots\ldots) * C_n + i_n - \text{lower bound effect.}$$

For example, the description of the declaration

<u>array</u> A, B $\left[1 : 3, 2 : 5\right]$ is:

$\langle 4 \rangle_{24} \langle \text{Not used} \rangle_{24}$

$\langle 12 \rangle_{24} \langle \text{Not used} \rangle_{24}$

STACK ADDRESS $\langle 6 \rangle_{24} \langle 2 \rangle_{24}$

$\langle \text{Description of B} \rangle_{48}$

$\langle \text{Description of A} \rangle_{48}$

10 CONSTANT $\langle x \rangle_3 \langle t \rangle_3 \langle \text{Stack Address} \rangle_{18} \langle 0 \rangle_1 \langle 10 \rangle_5 \langle \text{Not used} \rangle_{18}$

The Stack Address locates the constant in the stack.

11 EXPRESSION $\langle x \rangle_3 \langle t \rangle_3 \langle \text{Stack Reference} \rangle_{18} \langle 1 \rangle_1 \langle 11 \rangle_5 \langle \text{Segment Location} \rangle_{18}$

12 SIMPLE VARIABLE $\langle x \rangle_3 \langle t \rangle_3 \langle \text{Stack Address} \rangle_{18} \langle 0 \rangle_1 \langle 12 \rangle_5 \langle \text{Not used} \rangle_{18}$

The Stack Address locates the stack entry for the variable.

13 SUBSCRIPTED VARIABLE $\langle x \rangle_3 \langle t \rangle_3 \langle \text{Stack References} \rangle_{18} \langle 1 \rangle_1 \langle 13 \rangle_5 \langle \text{Segment Location} \rangle_{18}$

PROGRAM

```
ALGOL - 32    (1.0)                     DUMP
   DUMP    EXAMPLE
    ≠BEGIN≠
       ≠INTEGER≠ N .,
       ≠ARRAY≠ A1, A2(/1..3/) .,
       ≠PROCEDURE≠ BLOW(BA, IB, IC, ID, L, PA, AA, AB) .,
          ≠VALUE≠ ID, AB .,
          ≠INTEGER≠ IB, IC, ID .,
          ≠BOOLEAN≠ BA .,
          ≠LABEL≠ L .,
          ≠INTEGER≠ ≠PROCEDURE≠ PA .,
          ≠ARRAY≠ AA, AB .,
          ≠BEGIN≠
             ≠INTEGER≠ I .,
             ≠SWITCH≠ SW ..= L, LA .,
             I ..= PA ( I ) + 1 / 5 .,
             ≠FOR≠ I ..= 1 ≠STEP≠ 1 ≠UNTIL≠ 2 ≠DO≠ I ..= I + 1/0 .,
             ≠FOR≠ I ..= 2 ≠DO≠ .,
          LA..
         ≠END≠ .,
       ≠INTEGER≠ ≠PROCEDURE≠ K(I) ., ≠INTEGER≠ I ., K ..= I .,
       N ..= 4.7 .,
    BLOW ( ≠TRUE≠, N, 3.5+N, 2*N, L2, K, A1, A2 ) .,
    L2..
    ≠END≠        ≠EOP≠
```

DUMP

```
     CHANNEL,60=LU60,P80
     CHANNEL,61=LU61,P136,PP60
     CHANNEL,END
  ARITHMETIC OVERFLOW

          THIS ERROR OCCURED AFTER      LINE 0014

           IN THE BLOCK ENTERED AT      LINE 0003


   THE GLOBAL VARIABLES ARE ..
     UA,VALUE             03214   34 003335   01 406333
     UV                   03214   34 003335   01 406333
     LASTUSED             06361   14 600000   04 600001


   THE FORMAL VARIABLES ARE ..                                Explanation
                          06323   10 006237   02 001775   Return Information
                          06321   01 006301   10 000000   BA, Constant true
                          06317   03 006263   12 000000   IB, Simple Variable N
                          06315   14 006237   51 001757   IC, Expression 3.5+N
                          06313   00 000000   00 000012   ID, Value 2*N
                          06311   00 006237   04 001775   L , Label L2
                          06307   03 006237   46 001667   PA, Typed Procedure K
                          06305   02 006273   07 106255   AA, Array A1
                          06303   02 006355   07 106255   AB, Value Array, A2
                          06301   00 000001   00 000000       Value of constant true


   THE LOCAL VARIABLES ARE ..
                          06335   00 006327   04 001664   LA, Label
                          06337   00 006327   44 000011       for label not entered
                          06341   00 006327   04 001617       for label entered
                          06343   00 006327   04 001664       Designational Expression LA
                          06345   00 006237   04 001775       Designational Expression L
                          06347   00 000002   02 106345   SW, Switch
                          06351   00 000000   00 000001   I , Value of integer
                          06353   17 756314   63 146315       Working Location

        THIS BLOCK WAS CALLED FROM      LINE 0021

           IN THE BLOCK ENTERED AT      LINE 0001


   THE LOCAL VARIABLES ARE ..
                          06245   00 006237   04 001775   L2, Label
                          06247   03 006237   46 001667   K   Typed Procedure
                          06251   00 006237   45 001502   BLOW No type Procedure
                          06253   00 000003   21 003347       Dope Vector
                          06255   00 000001   00 000001       Dope Vector
                          06257   02 006265   07 106255   A2, Array
                          06261   02 006273   07 106255   A1, Array
                          06263   00 000000   00 000005   N , Value of integer

        THIS BLOCK WAS CALLED FROM      LINE 0001
```

CONTROL DATA CORPORATION
Documentation Department
3145 Porter Drive
Palo Alto, California

MAY 1966

PUB. NO. 60171000