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

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 OBJECT TIME ABNORMAL TERMINATION DUMP 3
Structured Dump 3
Global and Environmental Information 4

CHAPTER 3 GLOBAL INFORMATION 5
UA 5
UV 5
LASTUSED 5

CHAPTER 4 ENVIRONMENTAL INFORMATION 7
Formal Variables 7
Local Variables 7
Values of Variables 8
Descriptions of Variables 8

CHAPTER 5 DESCRIPTIONS 11
Terminology 11
02 Switch 11
03 String 12
04 Label or Designational Expression 12
05 No-type Procedure 12
06 Typed Procedure 12
07 Array 12
10 Constant 14
11 Expression 14
12 Simple Variable 14
13 Subscripted Variable 14

CHAPTER 6 SAMPLE PROGRAM AND DUMP 15
Program 15
Dump 16
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.
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.

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:

<table>
<thead>
<tr>
<th>Address Field</th>
<th>Information Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxxx</td>
<td>xx xxxxx xx xxxxx</td>
</tr>
</tbody>
</table>
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 . .

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
</table>
| UA, VALUE | xxxxx  
           | xx     
           | xxxxx  
           | xx     
           | xxxxx  |
| UV        | xxxxx  
           | xx     
           | xxxxx  
           | xx     
           | xxxxx  |
| LASTUSED  | xxxxx  
           | xx     
           | xxxxx  
           | xx     
           | xxxxx  |

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.
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
......     .....  
......     .....  
......     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:

**Boolean**

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

**Integer**

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

**Real**

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

DESCRIPTIONS OF VARIABLES

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

\(< x >_3 \ < t >_3 \ < \text{address 1} >_{18} \ < i >_1 \ < \text{kk} >_5 \ < \text{address 2} >_{18}\)

\(x\) 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 \(\text{address 1}\) and \(\text{address 2}\) 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:

<table>
<thead>
<tr>
<th>(x)</th>
<th>Possible Use</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No transformation</td>
<td>Formal and local</td>
</tr>
<tr>
<td>1</td>
<td>Fix</td>
<td>Formal only</td>
</tr>
<tr>
<td>2</td>
<td>Float</td>
<td>Formal only</td>
</tr>
<tr>
<td>3</td>
<td>Fix-then-float</td>
<td>Formal only</td>
</tr>
<tr>
<td>4-7</td>
<td>Not used</td>
<td>Not used</td>
</tr>
</tbody>
</table>
The type, t, can take the following values:

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Possible Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No type</td>
<td>Formal and local</td>
</tr>
<tr>
<td>1</td>
<td>Boolean</td>
<td>Formal and local</td>
</tr>
<tr>
<td>2</td>
<td>Real</td>
<td>Formal and local</td>
</tr>
<tr>
<td>3</td>
<td>Integer</td>
<td>Formal and local</td>
</tr>
<tr>
<td>4</td>
<td>Real-integer</td>
<td>Formal only</td>
</tr>
<tr>
<td>5</td>
<td>Integer-real</td>
<td>Formal only</td>
</tr>
<tr>
<td>6</td>
<td>Real-integer-real</td>
<td>Formal only</td>
</tr>
<tr>
<td>7</td>
<td>Not used</td>
<td>Not used</td>
</tr>
</tbody>
</table>

The kind, kk, can take the following values:

<table>
<thead>
<tr>
<th>Code</th>
<th>Kind</th>
<th>Possible Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Not used</td>
<td>Not used</td>
</tr>
<tr>
<td>01</td>
<td>Not used</td>
<td>Not used</td>
</tr>
<tr>
<td>02</td>
<td>Switch</td>
<td>Formal and local</td>
</tr>
<tr>
<td>03</td>
<td>String</td>
<td>Formal and local</td>
</tr>
<tr>
<td>04</td>
<td>Label or designational</td>
<td>Formal and local</td>
</tr>
<tr>
<td></td>
<td>expression</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>No-type procedure</td>
<td>Formal and local</td>
</tr>
<tr>
<td>06</td>
<td>Typed procedure</td>
<td>Formal and local</td>
</tr>
<tr>
<td>07</td>
<td>Array</td>
<td>Formal and local</td>
</tr>
<tr>
<td>10</td>
<td>Constant</td>
<td>Formal only</td>
</tr>
<tr>
<td>11</td>
<td>Expression</td>
<td>Formal only</td>
</tr>
<tr>
<td>12</td>
<td>Simple variable</td>
<td>Formal only</td>
</tr>
<tr>
<td>13</td>
<td>Subscripted variable</td>
<td>Formal only</td>
</tr>
<tr>
<td>14-37</td>
<td>Not used</td>
<td>Not used</td>
</tr>
</tbody>
</table>

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.
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:

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

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

\[\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\]

\[\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\]
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 0000011, 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.

own arrays are handled in the same way, except that their elements are assigned among the own 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

\[
\text{array } A \left[ l_1 : u_1, \quad l_2 : u_2, \quad \ldots \quad l_n : u_n \right]
\]

is:

\[
\begin{array}{cccc}
< C_n & >_{24} & < \text{Not used} & >_{24} \\
< C_{n-1} & >_{24} & < \text{Not used} & >_{24} \\
\ldots & \ldots & \ldots & \ldots \\
< C_2 & >_{24} & < \text{Not used} & >_{24} \\
< \text{Length of array} & >_{24} & < \text{Not used} & >_{24} \\
< \text{Lower bound effect} & >_{24} & < \text{No. of dimensions} & >_{24} \\
\end{array}
\]

where

\[
C_1 = u_1 - l_1 + 1
\]

Length of array = \( C_1 \times C_2 \times C_3 \times \ldots \times C_n \)

Lower bound effect = \( (( \ldots ( l_1 \times C_2 + l_2) \times C_3 + l_3) \times \ldots) \times C_n + l_n \)

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

\[
(( \ldots ( l_1 \times C_2 + i_2) \times C_3 + i_3) \times \ldots) \times C_n + i_n - \text{lower bound effect}.
\]
For example, the description of the declaration

\[
\text{array } A, B \left[ \begin{array}{c} 1 : 3, 2 : 5 \end{array} \right]
\]
is:

\[
\begin{array}{rcl}
<4 & >_{24} & \text{Not used}_{24} \\
<12 & >_{24} & \text{Not used}_{24}
\end{array}
\]

STACK ADDRESS

\[
\begin{array}{rcl}
<6 & >_{24} & 2 \\
< \text{Description of } B & >_{48} \\
< \text{Description of } A & >_{48}
\end{array}
\]

10 CONSTANT

\[
<x>_{3} <t>_{3} <\text{Stack Address}>_{18} <0>_{1} <10>_{5} <\text{Not used}>_{18}
\]
The Stack Address locates the constant in the stack.

11 EXPRESSION

\[
<x>_{3} <t>_{3} <\text{Stack Reference}>_{18} <1>_{1} <11>_{5} <\text{Segment Location}>_{18}
\]

12 SIMPLE VARIABLE

\[
<x>_{3} <t>_{3} <\text{Stack Address}>_{18} <0>_{1} <12>_{5} <\text{Not used}>_{18}
\]
The Stack Address locates the stack entry for the variable.

13 SUBSCRIPTED VARIABLE

\[
<x>_{3} <t>_{3} <\text{Stack References}>_{18} <1>_{1} <13>_{5} <\text{Segment Location}>_{18}
\]
SAMPLE PROGRAM AND DUMP

PROGRAM

ALGOL - 32  (1.0)  DUMP

0** 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 ,

10**

#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 ,

20**

N ..= 4.7 ,
BLOW ( #TRUE#, N, 3.5+N, 2*N, L2, K, A1, A2 ) ,
L2 .
#END#

#EOP#
DUMP

CHANNEL,60=LU60,P80
CHANNEL,bl=LU61,P13b,PP60
CHANNEL,END

ARITHMETIC OVERFLOW

THIS ERROR OCCURRED 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 ..
06323 10 00b237 02 001775 Return Information
06321 01 00e301 10 000000 PA, Constant true
06317 03 006263 12 000000 IB, Simple Variable N
06315 14 00b237 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 4b 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

16