SCALP
BLOCK DIAGRAMS

CCM-8A

Stephen J. Garland
Anthony W. Knapp
Thomas E. Kurtz

15 May 1963

Computation Center
Dartmouth College
Introduction

SCALP is an adaptation of ALGOL-30 for load-and-go operation. ALGOL-30 is a partial implementation of ALGOL-60 for the LGP-30, but which has not been publically released.

SCALP places these restrictions on ALGOL: (1) No blocks or procedures, except special functions; (2) No Boolean variables or operators; (3) No conditional expressions; (4) Only the while type for-statement; (5) No nested switch declarations, though nested switch calls are permitted; (6) Special integer divide not included. There are several symbol transliterations, and a number of machine-imposed limitations, but in all other general respects SCALP adheres to the form and spirit of ALGOL.

The reader is referred to CCM-7 (A Manual for SCALP) for detailed information on the external characteristics of SCALP and its operation.

This memorandum contains the block diagrams for SCALP. The coding sheets appear in a companion memorandum, CCM-8B, while the library of special functions is documented in CCM-8C.

In the absence of a detailed discussion of the block diagrams, it should be noted that the compiling follows closely the last-in, first-out method of translation proposed by Samelson and Bauer, and others. The symbol table follows the suggestion of E. J. Williams. These methods give extremely fast compiling speeds.

The principal work on SCALP was done by two students, Stephen J. Garland and Anthony W. Knapp. Garland did the compiler while Knapp composed the interpreter. Their work rests on the earlier work of ALGOL-30 which included also the efforts of Robert Hargreaves and Jorge Llacer.

The use of this material is permitted for any purpose provided appropriate credit is extended to Garland and Knapp.
### SCALP Instructions

<table>
<thead>
<tr>
<th>Tag</th>
<th>0</th>
<th>4</th>
<th>8</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>abs</td>
<td>sign</td>
<td>cos</td>
<td>sine</td>
</tr>
<tr>
<td>B</td>
<td>Bring</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>Subtract From</td>
<td>No operation</td>
<td>Subtract From</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Random</td>
<td>eq+</td>
<td>Relational</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>Divide into</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Divide</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>Multiply</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Matrix</td>
<td>Read Integer</td>
<td>Read Real</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>Print</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Power</td>
<td>entier</td>
<td>Power</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>Transfer</td>
<td>Switch</td>
<td>Exit</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Title</td>
<td>cr</td>
<td>tab</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>Trace Integer</td>
<td>Read Integer</td>
<td>Trace Read</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Float</td>
<td>Round</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Subtract</td>
<td>exp</td>
<td>Subtract</td>
<td></td>
</tr>
</tbody>
</table>

*on library tape.*
General Flow Diagram

Compile

Initialize

Read

at an AL-GOL symbol?

Store or in Number Collar - NC

Transfer according to a, last element of symbol Collar - SC

Transfer on a

Complete special instructions or declarations

Store a in SC

MComp

Complete instruction(s) according to a, ecess

Transfer again

Repeat

Done
Store

Trim accumulator to instruction address

acc. = 0?

Yes: y = address of accumulator

No: 2655

SUMP = acc. (store acc. in memory)

address portion of y = NHM

NUM = 2E42

Yes: Error, output STORE

No: 2465

NHM = NHM + 1

y = junk in location

End Store

The instruction being stored will be modified at no time by the instruction placed in location y.
Arithmetic Transfer

Store and Read

Check for Loading...

If $X > 0$ then...

Yes

No

$SC = SC + 1$

Yes

$SC = SC - 1$

Set Var. Flag positive

Read from Return

Read

Set Return to Read

Set Return to Repeat

Store and Read

$0 = 0$ or $0$

Yes

No

$2746$ Yes

$2747$ No

$2607$ No

Check $a + v$ unary $> 0$

$2510$

$2403$

$2927$

$2934$

$2737$

$2736$

$2735$
Array Headings

For $A[i,j]$, the following is compiled:

Compiled when $a = 0$, $SCA = [i]$:
- $\alpha$ is lower bound of 1st subscript
- $\sigma_{id}$ is stored to compile
- $\alpha = 0$  (no dimension of 2nd subscript)
- Address of $A[i]$ computed from $\sigma_{id}$

Compiled when $a = 1$, $SCH = E$:
- $\alpha = 0$  (no lower bound of 2nd subscript)
- $\sigma_{id}$ contains 1st loc of array = address stored in Phable
- $\sigma_{id}$ location to store address computed

KE sets up all constants for this compilation.

K subscript generates the instructions.
Unary Bring Generator, Store BTP, End Arithmetic

Unary Bring Gen 2625
acc = NCC 2627
Loop 2628
acc = 0? No 2746
Store BTP 2750
acc = BTP
Exit Unary BG 2761

Store BTP 2763
HTP < 0? No 2826
acc = HTP Store 2732
acc = B' BTP' 2780
Store 2790
No hold to store

End Arith 2603
TSC > TSL? No 2606
Error or No temp 3128

<EAH> = Type of BGT + TSC + 1

Store temp variable in NCC = set up in Bring Gen.

Type is real? Yes

Tag = real (2)

Tag = worst (1)

TSC = Tag TSC + 1 2016

HTP = Tag + H 'TSC'

Return 2017

Return 4 set either before the PComp transfer or on entry to the subroutine

Read

Return

Exit End Arith
Conditionals, Relational, For Statements

K (relational) = THEN or ELSE ?

Yes: SCC = SCC + 1

No: Error

K THEN

If = ELSE ?

Yes: NCC = NCC + 1

No: Error

K ELSE

NCC = NCC - 1

Add <NCC> = NNM (fill in U0000)

Repeat

K FOR

If = WHILE ?

Yes: Read From Repeat

No: Error

NCC = NCC + 1

<NCC> = NNM

Store and Read

These form the loop of the for statement

NCC = NCC - 2

Acc = U0000 + <NCC> + 17

Store

Add <NCC> = NNM

Repeat
Procedures

K()

Set Var. Flag -

if acc = 1 then
  if acc + 2 = 0 then
    print or read?
    if print then
      print 3301
    else
      if acc = 0 then
        error
      else
        read
      end if
    end if
  end if
end if

Unary
Bringing Gen

acc = proc. inst.

Adjust load code to local required routines

End Arith.

Store

acc = 0000

acc = HEV

Var Flag = -1

EAH = acc

Set return to Read Exit

acc = HEV

Store

Random

http or not?
Labels

K: NCC = NCC - 1

if NCC = 0 then
  type = undefined label?
  if type = no then
    Error label area
    ACC = 95000 + add acc<17> + 0.17
    Store
  else
    Store transfer to label in profile; used by previous 'goto's'
  end if
  Store U', M', I'
  ACC = ACC + 1157
  Input
  if Input = 0 then
    Read
    if <input > = 0 then
      Store U to profile element
    end if
  end if
end if

K: goto

XMI

if XMI = 3 then
  if a label or a switch is error label then
    Error label area
    ACC = <label - XMI>
  end if
  if a label is defined then
    Label defined
    ACC = U0000 + add acc<17> + 0.17
    Store
  end if
  Read
end if

Read
Declarations

1. **Declaration**: 3315

2. **Decl Type = occ** 3517

3. **AFlag = 0** 3548

4. **Input** 3610

5. **or an ALGOL symbol?** Yes 3651

6. **or a switch or label?** No 3796

7. **No** 3748

8. **Set up hold to store field identifiers**

9. **Yes** 3653

10. **NCC = NCC + 1** 3655

11. **List of array being declared kept in NC**

12. **List of transfers stored for switch**

13. **Decl Types Undefined Label** 3844

14. **Decl Types Undef Label** 3844

15. **S0 = ZL+1** 3607

16. **S0 = ZL+1 - 1**
Done, Load

Load

Read Code

Exit Load

Load Routine

No

Set up jump in interpreter
Transfer list from code

Storage available for routine?

Yes

Load Routine by
Relocatable Hex
with modifier = NHM + E
; react NHM

No

Scan tape past routine

i = i + 1

Load Routine

i = 1

i = 0

Load

Save

Print "Load"

Stop

Transfer

Print "Done"

Stop

Load switch

Normal

Print "Load"

Stop

Continue

End

Load Necessary?

Yes

Print "Error 4-bit" + Stop

No

F3
Memory Allocation

Indexing

Temporary Storage

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#C</td>
<td>etc</td>
</tr>
<tr>
<td>@C</td>
<td>Identifiers</td>
</tr>
<tr>
<td>TSC</td>
<td>Type</td>
</tr>
<tr>
<td>NCC</td>
<td>Value of Constant</td>
</tr>
<tr>
<td>SCC</td>
<td></td>
</tr>
<tr>
<td>ZCLIT</td>
<td></td>
</tr>
<tr>
<td>NHM</td>
<td>Program</td>
</tr>
</tbody>
</table>

Word Structure

- alpha-numeric identifier
- Type: on file loc, on entry loc
- Value of Constant
- Array: subscript to Array loc
- Switch
  - length of code switch loc

SCALP

- Instruction: bits 12-17
- Index: bits 5-6
- Address (add1): bits 10-15
- Add 2 = offset
  - bits 0-9
  - Type: bits 5, 6-5
  - Tag: bits 16-17 of inst
Symbols

A: acc -- accumulator; a temporary storage location; used as input to store

acid -- identification of last symbol read

AFlag -- flag which indicates whether Declaration is operating inside of normal pair list

AlO -- location of first array being declared in number cellar list

B: BGL -- Bring Generator Flag to indicate an array

BGt -- Bring Generator Temp.; contains type of result

BTP -- "Bring To Program" formation by Bring Generator

C: CT -- Comment Temporary; contains alphanumeric

D: D -- declare; e.g., Di means declare;

DefType -- defining type; set at "undefined label" except in Declarations

Diff -- difference between an Stable and Pthal location

E: EAH -- End Arithmetic Hold; usually set to store identifier for temporary variable in number cellar

EP -- End Program; contains exit instruction for program

Ertemp -- contains reset for NNM after an "Execute"

Ex2temp -- contains exit from an "Execute"
H:  HTP -- "Hold To Program"; matched against next BTP to see if both can be eliminated.

I:  IFlag -- used by interpreter on input to check type

K:  K -- "compile"; e.g., K+ means compile +

L:  LB -- lower bound of bound pair in array dec.

Load Code -- code word telling which routines on library tape are called for by program

loc to patch -- location of PS init. to be replaced by a UO to the patch

M:  Msize -- size of array being declared.

N:  NHM -- "next hold to memory" counter

O:  Op loc -- operand location in Bring Generator; i.e., location of y in x+y, etc.

P:  PTL -- "primary Table location" counter for stable reset

Prev -- Flag used to determine whether +, - is unary

S:  saved -- counter for storing UO instructions in switch declarations

Sch -- identifier of last entry in the symbol cellar

SSLo -- location of 1st subscript constant; used in array declarations
T: TCC - a) temporary #C counter
   b) used to record last value of #C to use
      for initializing in "Execute"

TSC -- temporary storage counter

U: UB -- upper bound of bound pair in array declaration

V: Var. Flag -- variable flag; tells whether last symbol
    read was a variable or not

<x > -- contents of location x; e.g., <NCC> =
 contents of the location whose address is
 given by NCC

<< x >> -- iteration of < >; contents of the location
 whose address is in location x
Symbol Types

ALGOL Symbols:

Group 1: Tag = -103
then, while, do, else, end, :, ), ), ),

Group 2: Tag = -203 + n05
n = 0 ;

n = 1 =

n = 2 +

n = 3 , , , , = , >

Group 3: Tag = -303
, blank, array, read, switch, integer, comment, title,
quote, go to, c tab, escrs, =, random, for, E
Tag = -303 + le5 : print, read

Group 4: Tag = -403 + le5
begin, if, ; =, (x)

Group 5: Tag = -503
sin, cos, log, exp, sqrt, arctan, abs, sign, either

Variables:

le1 - label
le2 - not used
le3 - array (switch with le1)
le4 - array subscript flag (+1 only in Nc)
le5 - real flag; 0 mean integer
Each routine should have an even multiple of 6 instructions, and is punched by the ALGOL relocatable hex punch routine. Immediately preceding each routine is a code word telling:

1) the location of the subroutine call (in track 03) @ q=29
2) (n-1) @ q=n, where n = no. locations required

Tape format: 8222 8222
{ code }
{ routine } repeated 10 times

80000000

Order of routines:  EO
    ln
    exp
    Ej
    sin-cos
    arctan
    sqrt
    random
    enter