Implementing EM: Dependency as agency

Ashley Ward
November 2003

Session outline

- Review of the von Neumann machine
- What is dependency?
- Dependency as agency
  - Eden
  - ADM
  - SCICS
  - DAM
- Generalisation
  - Evaluation / Storage strategies
  - O/C-U coordination

State in a von Neumann machine

Code interpretation	Raw memory data: Address	Raw memory data: Data	Value interpretation
-	000091FC	65480000	00 00 H e
-	00009200	206F6C6C	l l o (space)
-	00009204	6C726F57	W o r l
(ALIGN)	00009208	00000064	d - - -
STMFD R13!,{R0-R12,R14}	0000920C	E92D5FFF	- - - -
MOV R0,#65	00009210	E3A00041	- - - -
STRB R0,91FC	00009214	E54F0020	- - - -
LDRB R1,91FC	00009218	E55F1024	- - - -
ADD R1,R1,#2	0000921C	E2811002	- - - -
STRB R1,91FD	00009220	E54F102B	- - - -
LDRB R0,91FD	00009224	E55F002F	- - - -
SWI 0	00009228	EF000000	- - - -
ADR R1,91FE	0000922C	E24F1036	- - - -
MOV R2,#0	00009230	E3A02000	- - - -
LDR R0,{R1,R2}	00009234	E7910002	- - - -
SWI 0	00009238	EF000000	- - - -
ADD R2,R2,#1	0000923C	E2822001	- - - -
CMP R2,#10	00009240	E352000A	- - - -
BLE 9234	00009244	DAFFFFFA	- - - -
LDMFD R13!,{R0-R12,R15}^	00009248	E8FD9FFF	- - - -

Characteristic properties of a von Neumann machine

1. Instructions and data are defined only implicitly through usage
2. There is a single memory, which is sequentially addressed
3. Memory is one-dimensional
4. The "meaning" (type) of data is not stored with it

Glenford Myers (1982) Advances in Computer Architecture

2

Defects in procedural languages

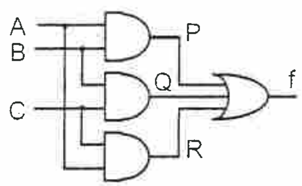
- Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak:
  - their primitive word-at-a-time style of programming inherited from their common ancestor - the von Neumann computer,
  - their close coupling of semantics to state transitions,
  - their division of programming into a world of expressions and a world of statements,
  - their inability to effectively use powerful combining forms for building new programs from existing ones,
  - and their lack of useful mathematical properties for reasoning about programs.

John Backus Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs 1977 ACM Turing Award Lecture

Dependency

a *is* b + c

Dependency in logic circuits



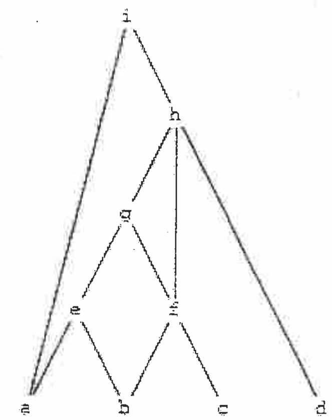
A	B	C	P	Q	R	f
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	1
1	0	0	0	0	0	0
1	0	1	0	0	1	1
1	1	0	1	0	0	1
1	1	1	1	1	1	1

```
P is A and B;
Q is B and C;
R is A and C;
f is P or Q or R;

f is (A and B) or
     (B and C) or
     (A and C);
```


The script graph

```
a = 3
b = 10
c = 7
d = 9
e = add(a,b)
f = add(b,c)
g = times(e,f)
h = max3(g,f,d)
i = power(h,a)
```

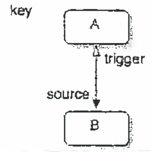


R. I. Cartwright. Geometric Aspects of Empirical Modelling: Issues in Design and Implementation. Ph.D thesis, University of Warwick, September 1998.

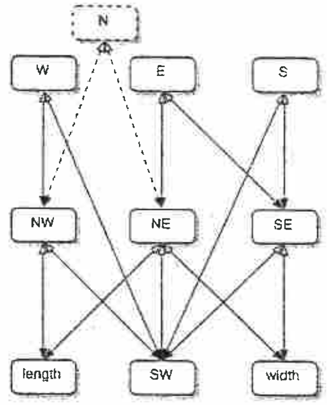
Extending a script graph

donald script with extension

```
%donald
line W, E, S
point SE, SW, NE, NW
int length, width
W = [NW, SW]
E = [NE, SE]
S = [SW, SE]
SE = SW + {width, 0}
SW = {100, 100}
NE = SW + {width, length}
NW = SW + {0, length}
length = 800
width = 800
line N
N = [NW, NE]
```



dependency graph with extension



What exactly is dependency?

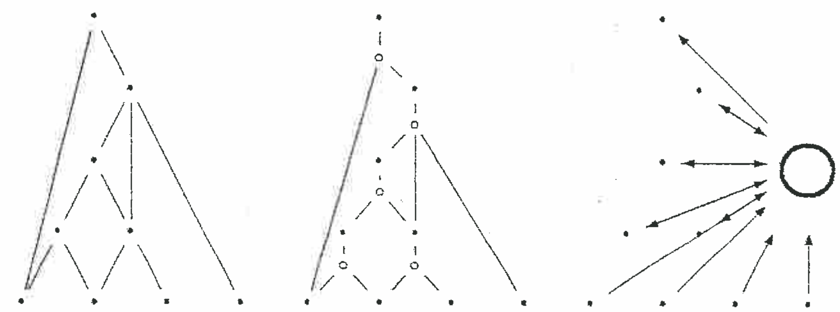
- A dependency is a relationship between observables that pertains in the view of a particular agent.
  - It expresses the empirically established fact that when the value of a particular observable x is changed, other observables (the dependants of x) are
    - » of necessity
    - » changed in a predictable manner
    - » as if in one and the same action.
  - The changes to the values of x and its dependants are indivisible in the view of the agent.
  - That is: no action or observation on the part of the agent can take place in a context in which x has changed, but the dependants of x have yet to be changed.

Interactive Situation Models for Information Systems Development by W.M.Beynon, R. Cartwright, P-H. Sun and A. Ward. Proceedings of SCI'99 and ISAS'99, Volume 2, pp9-16, Orlando, USA July 1999.

Dependency as restrictions on observation and/or action

- Restrictions on observation on the part of the perceiving agent
  - Observation is not possible during updating agent action
  - No observation is made during updating agent action
  - Due to relative speeds, perceiving agent cannot observe updating agent action
- Restrictions on action on the part of the perceiving agent
  - Action is not possible during updating agent action
  - Action is ignored or buffered during updating agent action
  - Required timing of agent action is too fine for it to be accurately possible

Dependency as agency



Dependency as agency in Eden

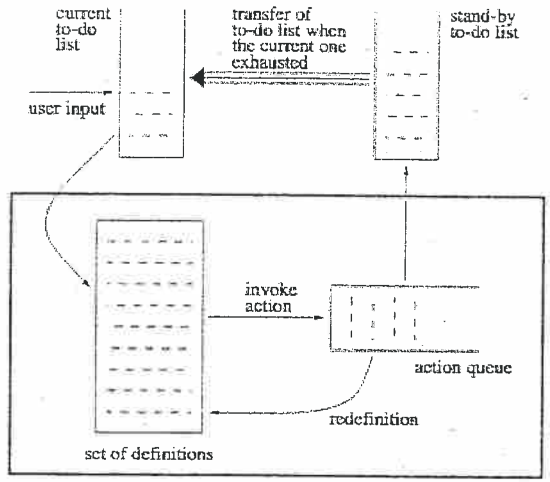
Dependency: `a is b+c;`

Agency: `proc updatea : b,c {a = b+c;}`

In the action-based version:

- There is no easily identifiable "location" for the definition of a
  - » Harder to debug a script
  - » Possibility of more than one definition of a, causing interference
- Current Eden implementation doesn't "know" what is written to when an action is run
  - » There is no check for cyclic dependency
  - » Update ordering could be less efficient
- The action could read from variables not specified as a trigger
  - » Creates cases of missing dependency

Eden execution



Simon Yung (July 1996) EPSRC report

Eden agency control "hacks"

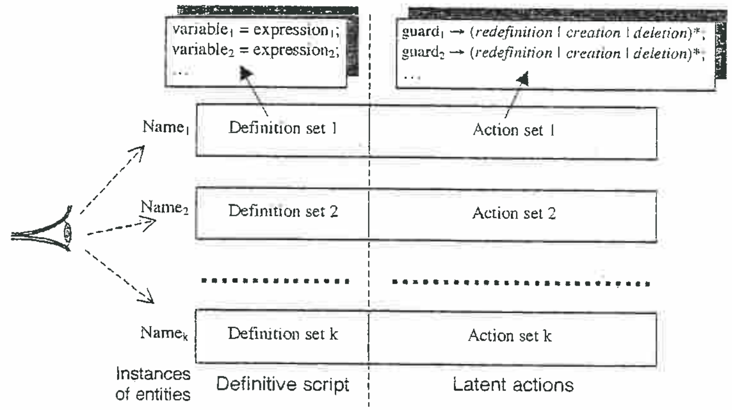
- todo(script_string);
  - executes script_string along with user input after all the current actions have finished executing
  - typical use: for a clocking mechanism
- autocalc = 0; (also autocalc = 1;)
  - controls "recalculation"
  - typical use: to form blocks of redefinitions
- eager();
  - forces actions to be performed and screen to be updated, even in the middle of a for loop
  - typical use: to show animation
- procmacro
  - marks a block of redefinitions, like "proc", but effectively with an eager(); automatically inserted between each statement
  - typical use: to replay user interaction

Rationalisation proposal: the 4-layer prioritised action system

- Definition layer
  - Maintains state specified by definitions
- Meta-definition layer
  - Maintains Higher Order Definitions (HODs)
- Decision layer
  - "... determines what actions are needed to transit the state"
- Transition layer
  - "... performs the actions determined by the decision layer"
- "... if we have prioritised actions, we can in principle remove definitions altogether. Of course, we still want to keep the definitional form for clarity".

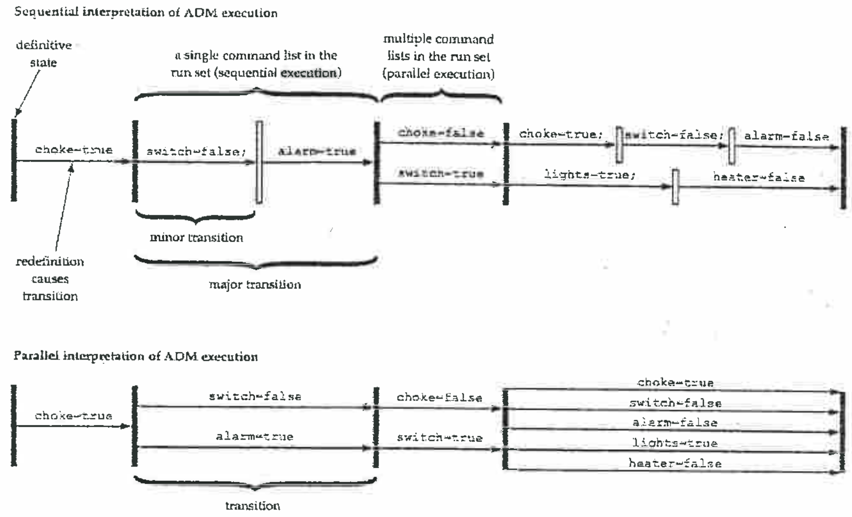
Simon Yung (July 1996) EPSRC report

The ADM



Jararatsri Rungrattanaubol. *A treatise on Modelling with definitive scripts*. PhD thesis, University of Warwick, April 2002. Fig 3-4

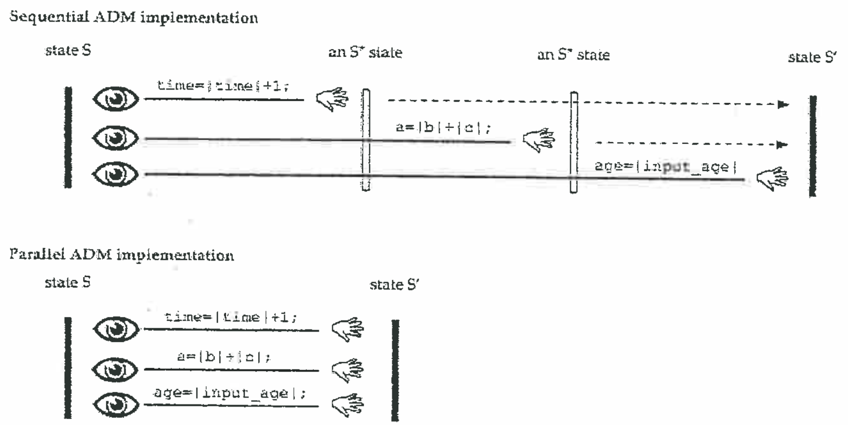
ADM execution



ADM implementations

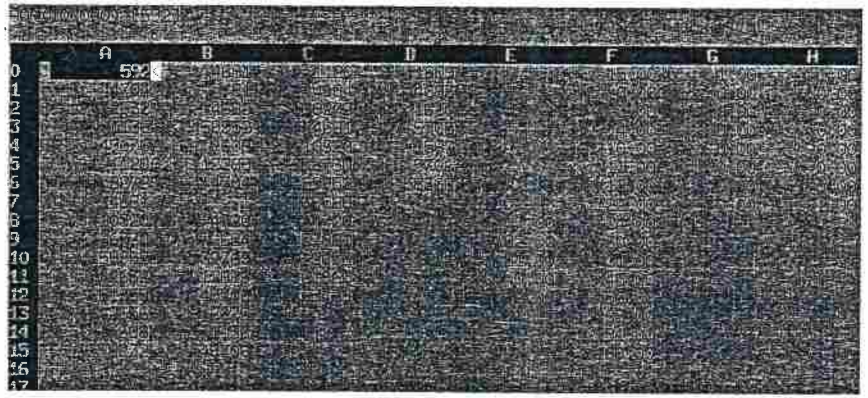
- am (Mike Slade, 1989)
  - "A simulator, in the sense that the commands in the run set are performed sequentially, rather than in parallel"
  - Written in C and uses lex and yacc
  - 5000 LOC
- adm, adm2, adm3
  (Simon Yung 1992 & 1996, Pi-Hwa Sun 1999)
  - Translators from ADM to Eden

am implementation

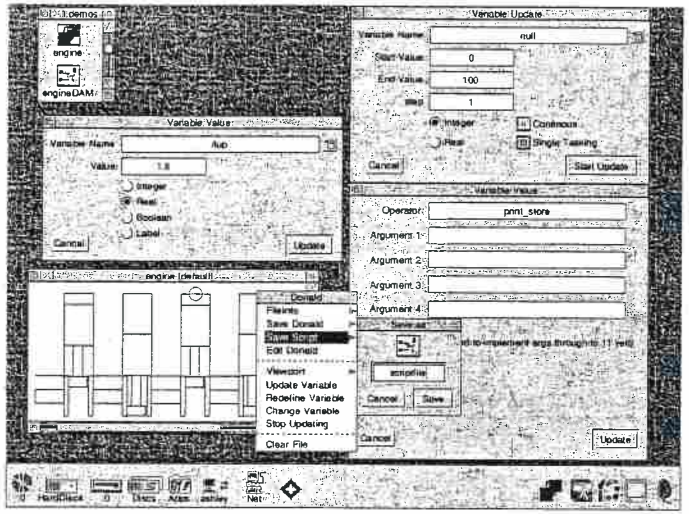


scicsYung1995

SCICS = sc (Spreadsheet Calculator) + ics ("Introduction to Computer Science" machine simulator)



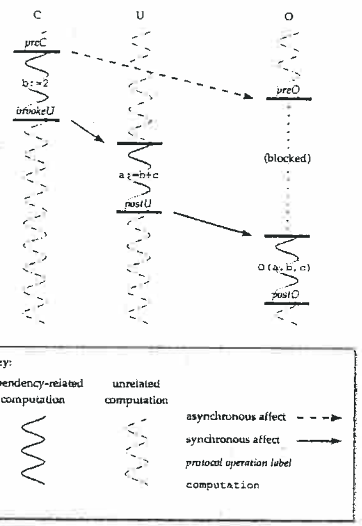
DAM demo



Evaluation / Storage strategies

1. evaluate at every use
   » storing only formulae
2. evaluate at every redefinition
   » storing formulae and values
3. a mix of 1 and 2: evaluate at use when a redefinition has previously out-dated the store
   » storing formulae, values and out-of-date flags

Observation/Change-Update coordination



Listing 4.2 shows an interaction with am<footnote: on line 2, cat is a standard UNIX command, not a feline> that uses the script in Listing 4.1. User input is shown in bold.

```
entity flap() {
definition
      lflap = 5,          # the length of the flap
      angle = 0,          # inclination of the flap
      switch = true,      # whether the electronic lock
is operating
      fourWayLock = 0,    # 1: cat can go out only, 2: in
only
                          # 0: no restriction, 4: no
access
      Radius = 10,        # electronic lock detector
range
      elecLock = (pos > Radius || pos < -Radius) &&
switch,
      canPushOut = angle != 0 || (!elecLock &&
(fourWayLock == 0 ||
fourWayLock == 1)),
      canPushIn = angle != 0 || (!elecLock &&
(fourWayLock == 0 ||
fourWayLock == 2))
action
      pushOut && canPushOut
print("Angle becomes ", angle+1)
            -> angle = |angle| + 1,
      pushIn && canPushIn
print("Angle becomes ", angle-1)
            -> angle = |angle| - 1,
      !pushOut && !pushIn && angle > 0
print("Angle becomes ", angle-1)
            -> angle = |angle| - 1,
      !pushOut && !pushIn && angle < 0
print("Angle becomes ", angle+1)
            -> angle = |angle| + 1
}


entity man() {
definition
      doit = 0
action
      doit == 1 && switch == false
print("Switch on")
            -> switch = true,
      doit == 11 && switch == true
print("Switch off")
            -> switch = false,
      doit == 21 && angle == 0 && fourWayLock != 0
print("Four way lock is set to 0")
            -> fourWayLock = 0,
      doit == 31 && angle == 0 && fourWayLock != 1
print("Four way lock is set to 1")
            -> fourWayLock = 1,
      doit == 41 && angle == 0 && fourWayLock != 2
print("Four way lock is set to 2")
            -> fourWayLock = 2,
      doit == 51 && angle == 0 && fourWayLock != 3
print("Four way lock is set to 3")
            -> fourWayLock = 3,
      true -> doit = |rand(60)|
}

entity cat() {
definition
      height = 2,         # height of the cat
      pos = 2,            # >0: outside the house, <0:
inside
      obstruct = 0,
      intention = -1,     # 1: going out, -1 coming in, 0
stay put
      pushOut = intention > 0 && obstruct,
      pushIn = intention < 0 && obstruct
action
      intention > 0 && !obstruct
print("Cat moves forward")
            -> pos = |pos| + 1,
      intention < 0 && !obstruct
print("Cat moves backward")
            -> pos = |pos| - 1
}
```

Listing 4.1: Simon Yung's am catflap script

```
$ cd ~empublic/projects/catflapYung1994
$ cat flap-noinstantiate.am - |
/dcs/emp/empublic/bin/am-1.1
am-1.1> compiling flap()
am-1.1> compiling man()
am-1.1> compiling cat()
am-1.1> l en                # list entity descriptions (P)

            ENTITY LIST
            ***********
entity flap() {             (0 parameters)
...

entity cat() {              (0 parameters)
DEFINITION
      height = 2,
      pos = 2,
      obstruct = 0,
      intention = -1,
      pushOut = intention>0&&obstruct,
      pushIn = intention<0&&obstruct
ACTION
      intention>0&&!obstruct print("Cat moves forward")
->
            pos = |pos|+1,

      intention<0&&!obstruct print("Cat moves
backward") ->
            pos = |pos|-1
}
0 instances
            END OF ENTITY LIST
            ******************
am-1.1> l in                # list instances
            INSTANCES
            *********
            END OF INSTANCES
am-1.1> l ds                # list definition store (D)

            DEFINITION STORE
            ****************
            END OF DEFINITION STORE
            ***********************
am-1.1> l as                # list action store (A)

            ACTION STORE
            ************
            END OF ACTION STORE
            *******************
am-1.1> cat ()              # instantiate new cat entity
instantiating cat
am-1.1> l in
            INSTANCES
            *********
cat ()
            END OF INSTANCES
am-1.1> l ds

            DEFINITION STORE
            ****************
Variable # 1: height = 2
Variable # 2: pos = 2
Variable # 3: obstruct = 0
Variable # 4: intention = -1
Variable # 5: pushOut = intention>0&&obstruct
Variable # 6: pushIn = intention<0&&obstruct
            END OF DEFINITION STORE
            ***********************
am-1.1> l as

            ACTION STORE
            ************
Action # 1: intention>0&&!obstruct print("Cat moves
forward") ->
            pos = |pos|+1
Action # 2: intention<0&&!obstruct print("Cat moves
backward") ->
            pos = |pos|-1
            END OF ACTION STORE
            *******************
am-1.1> ?(pos)              # show current definition and
value
pos is defined as 2
pos evaluates to 2
```

```
am-1.1> set iterations = 4     # limit to 4 major
transitions only
am-1.1> start              # start the machine executing
starting simulation
#
 Cat moves backward
#
 Cat moves backward
#
 Cat moves backward
#
 Cat moves backward
* 4 iterations successfully completed
am-1.1> ?(pos)             # cat is now inside the house
pos is defined as -1-1
pos evaluates to -2
am-1.1> define intention = 1;  # cat now wants to go out
defining intention
am-1.1> flap()             # instantiate the catflap
entity
instantiating flap
am-1.1> define obstruct = pos == 0 && angle < 3;
                           # ideally this is defined in
terms of
                           # lflap * tan(angle), pos and
height,
                           # but am does not have tan()
defining obstruct
am-1.1> l ds               # we now have definitions from
flap

            DEFINITION STORE
            ****************
Variable # 1: height = 2
Variable # 2: pos = -1-1
Variable # 3: obstruct = pos==0&&angle<3
Variable # 4: intention = 1
Variable # 5: pushOut = intention>0&&obstruct
Variable # 6: pushIn = intention<0&&obstruct
Variable # 7: lflap = 5
Variable # 8: angle = 0
Variable # 9: switch = TRUE
Variable # 10: fourWayLock = 0
Variable # 11: Radius = 10
Variable # 12: elecLock = (pos>Radius||pos<-
Radius)&&switch
Variable # 13: canPushOut =
angle!=0||(!elecLock&&(fourWayLock==0||fourWayLock==1))
Variable # 14: canPushIn =
angle!=0||(!elecLock&&(fourWayLock==0||fourWayLock==2))
            END OF DEFINITION STORE
            ***********************
am-1.1> set iterations = 8
am-1.1> start              # output from major transitions
are separated by '#'
                           # characters.  Note the flap
descends
                           # in parallel with the cat
motion
starting simulation
#
 Cat moves forward
#
 Cat moves forward
#
 Angle becomes 1
#
 Angle becomes 2
#
 Angle becomes 3
#
 Cat moves forward
 Angle becomes 2
#
 Cat moves forward
 Angle becomes 1
#
 Cat moves forward
 Angle becomes 0
* 8 iterations successfully completed
am-1.1> man()
instantiating man
am-1.1> start              # man randomly acts on the flap
starting simulation
```

```
#
 Cat moves forward
#
 Cat moves forward
#
 Cat moves forward
 Four way lock is set to 1
#
 Cat moves forward
#
 Cat moves forward
#
 Cat moves forward
#
 Cat moves forward
 Switch off
#
 Cat moves forward
* 8 iterations successfully completed
am-1.1> ^D          # exit am
$
```

Listing 4.2: An interaction with am

```
@0:110
+7
@0:000

LDA 1 120
STA 1 140        ~G0 = F0

~ @0:002
~ setup heap
LDX 1 110        ~ X=i
LDA 2 160        ~ H[X]
JCP 1 014
LDA 2 140        ~ G[X]
ADA 0 120        ~ &F[G[X]]
STA 1 111
ADX 0 120        ~ &F[X]
STX 1 112
SBR 1 060        ~ call swap
LDX 1 110
DXT 1 003

~ @0:015
LDX 1 140        ~X = G0
LDA 1 140        ~G0
ADA 0 120        ~&F[G0]
STA 1 111
LDA 0 121        ~&F1
STA 1 112
SBR 1 060        ~call swap
LDA 1 140
SBA 0 001
STA 1 140        ~G0--
LDA 0 001
STA 1 160        ~ H0 = 1

~ @0:031
LDA 1 160
SLL 0 001        ~ 2 * H0
SBA 1 140        ~ 2 * H0 - G0
JCP 1 052

~ @0:035
LDX 1 160
LDA 2 160        ~ H[H0]
JCP 1 052
LDA 2 140
STA 1 114        ~G[H0]
ADA 0 120        ~F[G[H0]]
STA 1 111
ADX 0 120        ~F[H0]
STX 1 112
SBR 1 060        ~swap
LDA 1 114
STA 1 160
JMP 1 031

~ @0:052
LDX 1 140
DXT 1 015

HLT

~ swap 111 & 112 using 113 as buffer
@0:060
LDA 3 111
STA 1 113
LDA 3 112
STA 3 111
LDA 1 113
STA 3 112
JMP 2 0

%10000
END
```

```
# This data file was
generated by the
Spreadsheet Calculator.
# You almost certainly
shouldn't edit it.

format A 10 0 0
format B 10 0 0
format C 10 0 0
format D 10 0 0
let A0 = 592
let A1 = 4704
let A2 = 2632
let A3 = 1136
let A4 = 49676
let A5 = 1120
let A6 = 8272
let A7 = 4681
let A8 = 10320
let A9 = 6730
let A10 = 59952
let A11 = 2632
let A12 = 61955
let A13 = 2656
let A14 = 608
let A15 = 8272
let B0 = 4681
let B1 = 81
let B2 = 4682
let B3 = 59952
let B4 = 608
let B5 = 12289
let B6 = 4704
let B7 = 1
let B8 = 4720
let B9 = 624
let B10 = 32769
let B11 = 12896
let B12 = 49706
let B13 = 2672
let B14 = 1136
let B15 = 49706
let C0 = 1120
let C1 = 4684
let C10 = 2656
let C11 = 61965
let C12 = 63488
let C2 = 8272
let C3 = 4681
let C4 = 10320
let C5 = 6730
let C6 = 59952
let C7 = 588
let C8 = 4720
let C9 = 57881
let D0 = 1609
let D1 = 4683
let D2 = 1610
let D3 = 5705
let D4 = 587
let D5 = 5706
let D6 = 58368
let E8 = 7
let F0 = 15
let F1 = 11
let F2 = 12
let F3 = 10
let F4 = 100
let F5 = 45
let F6 = 30
let F7 = 31
let F8 = 91
let F9 = 0
let F10 = 22
let F11 = 83
let F12 = 4
let F13 = 56
let F14 = 3
let F15 = 24
let G0 = 15
let G1 =
G0<2?0:(G0<3?2:(F2>F3?2:
3))
let G2 =
G0<4?0:(G0<5?4:(F4>F5?4:
5))
let G3 =
G0<6?0:(G0<7?6:(F6>F7?6:
7))
let G4 =
G0<8?0:(G0<9?8:(F8>F9?8:
9))
let G5 =
G0<10?0:(G0<11?10:(F10>F
11?10:11))
let G6 =
G0<12?0:(G0<13?12:(F12>F
13?12:13))
let G7 =
G0<14?0:(G0<15?14:(F14>F
15?14:15))
let H1 =
F1>@nval("F",G1)?1:0
let H2 =
F2>@nval("F",G2)?1:0
let H3 =
F3>@nval("F",G3)?1:0
let H4 =
F4>@nval("F",G4)?1:0
let H5 =
F5>@nval("F",G5)?1:0
let H6 =
F6>@nval("F",G6)?1:0
let H7 =
F7>@nval("F",G7)?1:0
```