S2 microsteps

opcode ld st jmp jal jr add sub mul div and or xor shl shr trap
addressing mode
a absolute
d displacement
x index
i immediate

unique opcode names
lda ld r1,temp
ldd ld r1,10(r2)
ldx ld r1,(r2+r3)
ldi ld r1,#100
add add r1,r2,r3
addi add r1,#20
others are similar
sta std stx sub subi mul muli div divi and andi or ori xor xori
shl shr trap
(note: no sti, shli, shri)

instruction format
L-format op:5 r1:5 ads:22
D-format op:5 r1:5 r2:5 disp:17
X-format op:5 r1:5 r2:5 r3:5 xop:12

<FETCH>
MAR = PC
MDR = M[MAR]; PC = PC+1

<DECODE>
IR = MDR

<LDA>
MAR = IR:ADS
MDR = M[MAR]
R[IR:R1] = MDR

<LDD>
T = R[IR:R2] + IR:DISP
MAR = T
MDR = M[MAR]
R[IR:R1] = MDR

<LDX>
T = R[IR:R2] + R[IR:R3]
MAR = T
MDR = M[MAR]
R[IR:R1] = MDR

<LDI>
R[IR:R1] = IR:DISP

<STA>
MAR = IR:ADS
MDR = R[IR:R1]
M[MAR] = MDR

<STD>
T = R[IR:R2] + IR:DISP
MAR = T
MDR = R[IR:R1]
M[MAR] = MDR

<STX>
T = R[IR:R2] + R[IR:R3]
MAR = T
MDR = R[IR:R1]
M[MAR] = MDR

<JMP>
IF TESTCC(IR:R1) PC = IR:ADS

<JAL>
R[IR:R1] = PC
PC = IR:ADS

<JR>
PC = R[IR:R1]

<ADD>
T = R[IR:R2] + R[IR:R3]
R[IR:R1] = T

<ADDI>
T = R[IR:R2] + IR:DISP
R[IR:R1] = T

similary for SUB SUBI MUL MULI DIV DIVI AND ANDI OR ORI XOR XORI

<SHL>
T = SHIFLEFT(R[IR:R2])
R[IR:R1] = T

<SHR>
T = SHIFTRIGHT(R[IR:R2])
R[IR:R1] = T

<TRAP>
simulator dependent

timing clocks (assume no wait memory)
lda 6
ldd 7
ldx 7
ldi 4
sta 6
std 7
stx 7
jmp 4
jal 5
jr 4
add 5
addi 5
. . .
shl 5
shr 5
trap 4

==========
S2 (a 32-bit version of S1)

S2 is a 32-bit version of S1, in other words, it is a S1 on steroid. The main aim of of S2 design is to increase the address space. The address space of S1 is 10 bits hence 1024 words which is not enough to run any realistic program. In order not to introduce too many changes into S2, most instructions in S2 follows S1 ISA with extension. There are enough bit in an instruction to encode three addresses, therefore 3-address instruction format is adopted. The instruction set is more "complete" such as full integer arithmetic and logic to enable compiling a realistic program into this ISA. Three new addressing modes are introduced: displacement, index and immediate. These mode enables a more realistic access to data structure such as a local variable in an activation record and array indexing. To sum up, S2 ISA represents a realistic ISA for a typical 32-bit processor.

This is not "optimal" design. There are plenty of room for improvement.

Instruction format

L-format op:5 rd1:5 ads:22
D-format op:5 rd1:5 rs2:5 disp:17
X-format op:5 rd1:5 rs2:5 rs3:5 xop:12

(rd dest, rs source, ads and disp sign extended)

L-format is similar to S1. D-format is new. X-format resembles S-format of S1.

Instructions are fixed length at 32-bit. Register set is 32, with R[0] always return zero. The address space is 32-bit, addressing is word. Flags are: Z zero, S sign, C carry, O overflow/underflow.

ISA and opcode encoding
mode: a - absolute, d - displacement, x - index, i - immediate, r - register
r2 - register 2 operands, s - special - 1 operand

opcode op mode format
0 ld a L
1 ld d D
2 ld i L
3 st a L
4 st d D
5 jmp a L use r1 as condition
6 jal a L
7 add i D
8 sub i D
9 mul i D
10 div i D
11 and i D
12 or i D
13 xor i D
14..30 undefined
31 xop - X

jump conditional coding in r1:
0 always, 1 eq, 2 neq, 3 lt, 4 le, 5 ge, 6 gt

xop

```
0 add       r           X
1 sub       r           X
2 mul       r           X
3 div       r           X
4 and       r           X
5 or        r           X
6 xor       r           X
7 shl       r2          X
8 shr       r2          X
9 ld        x           X
10 st       x           X
11 jr       s           X  use r1
12 trap s  X  use r1 as number of trap
13..4095 undefined

Meaning
format: op dest, source1, source2
r0 always returns zero

ld r1,ads  R[r1] = M[ads]
ld r1,#n    R[r1] = n
ld r1,d(r2) R[r1] = M[ d + R[r2] ]
ld r1,(r2+r3)           R[r1] = M[ R[r2] + R
[r3] ]
st ads,r1  M[ads] = R[r1]
st d(r2),r1             M[ d + R[r2] ] = R
[r1]
st (r2+r3),r1           M[ R[r2] + R[r3] ] =
R[r1]
jmp cond,ads            if cond true PC =
ads
  cond: always, eq Z=0, neq Z=1, lt S=1,
le S=1 or Z=1, ge S=0 or Z=1, gt S=0
jal r1,ads  R[r1] = PC; PC = ads
            // jump and link, to subroutine
jr  r1                  PC = R[r1]
            // return from subroutine

arithmetic
two-complement integer arithmetic

add r1,r2,r3            R[r1] = R[r2] + R[r3]
add r1,r2,#n            R[r1] = R[r2] + sign
extended n (n is 17 bits)
...

add, sub affect Z,C -- C indicates carry
(add) or borrow (sub)
mul, div affect Z,O -- O indicates
overlow (mul) or underflow (div) and
divide by zero

logic (bitwise)
and r1,r2,r3            R[r1] = R[r2] bitand
R[r3]
and r1,r2,#n            R[r1] = R[r2] bitand
sign extended n
or xor      . . .
shl r1,r2   R[r1] = R[r2] shift left one bit
shr r1,r2   R[r1] = R[r2] shift right one bit

affect Z,S bits

trap n                  special instruction,
n is in r1-field.

The opcode format and assembly
language format for S2 follow the
tradition dest = source1 op source2 from
PDP, VAX, 360. As r0 always is zero,
many instructions can be synthesis using
r0.
or r1,r2,r0 move r1 <- r2
or r1,r0,r0 clear r1
sub r0,r1,r2    compare r1 r2  affects
flags

To complement a register, xor with
0xFFFFFFFF (-1) can be used.
xor r1,r2,#-1           r1 = complement r2

28 November 2001
Prabhas Chongstitvatana
========================
;;  test assembly file for as2
.s
print 1        ;; special trap to print R[0]
stop 0
temp 101
one 102
array 103
base 5
idx 6
.a 10
.c
:start      ld r1 one  ;; absolute address
            ld r2 @10 r3            ;;
displacement address
            ld r1 +base idx         ;; index
address
            ld r1 #temp
            st array r1
            st @one base r1
            st +r2 r3 r1
            jmp eq loop
            jal r3 sub1
            jmp always start  ;; jump
uncondition
:loop       add r1 r3 #4            ;;
immediate address
            add r1 r2 r3
            or r1 r2 r0 ;; move r1 <- r2
            xor r1 r2 #-1           ;; r1 =
complement r2
            mul r1 r2 r3
:sub1       shl r3 r4
            jr r3                   ;; return
from subroutine
            trap print
            trap stop
:data                               ;; data
segment
.w 11 22 33                         ;; size
word
.e
==================

// matrix multiplication

//  N = 4, simulate matrix by (i,j) = i*N + j
N, a[16], b[16], c[16];

main() {
  N = 4;
  inita();
  initb();
  matmul();
  show();
}

inita() {
  i = 0;
  while( i<N ){
    j = 0;
    while( j<N ){
      a[index(i,j)] = i+1;
      j = j + 1;
    }
    i = i + 1;
  }
}

initb() {
  i = 0;
  while( i<N ){
    j = 0;
    while( j<N ){
      b[index(i,j)] = j+1;
      j = j + 1;
    }
    i = i + 1;
  }
}

index(i,j){
  return i*N + j;
}

matmul() {
  i = 0;
  while( i<N ){
    j = 0;
    while( j<N ){
      s = 0;
      k = 0;
      while( k<N ){
        s = s + a[index(i,k)] * b[index
(k,j)];
        k = k + 1;
      }
      c[index(i,j)] = s;
      j = j + 1;
    }
    i = i + 1;
  }
}

show() {
  i = 0;
  while( i<N ){
    j = 0;
    while( j<N ){
      print(c[index(i,j)], " ");
      j = j + 1;
    }
    print("\n");
    i = i + 1;
  }
}
======================
.s
 rz 0
 retv 28
 bp 29
 rads 30
 sp 31
 print 1
 printch 2
.a 0
.c
```

```
    ld sp #1000 ;; initialise sp
    ld bp #3000 ;; initialise bp
    ;; ---- Call 5
    jal  rads label_5
    ;; ---- Stop
    trap  stop rz
    ;; ---- Func 0 0
  :label_5
    add  bp bp #1
    st   @0 bp rads
    ;; ---- save regs
    st   @1 sp r1
    add  sp sp #1
    ;; ---- Lvalg 1
    ;; ---- Lit 4
    ;; ---- Set
    ;; ---- 7:   N = 4;
    ld   r1 #4
    st   @2001 rz r1
    ;; ---- Call 30
    jal  rads label_30
    ;; ---- Call 119
    jal  rads label_119
    ;; ---- Call 225
    jal  rads label_225
    ;; ---- Call 385
    jal  rads label_385
    ;; ---- Ret0
    ld   rads @0 bp
    sub  bp bp #1
    ;; ---- restore regs
    ld   r1 @0 sp
    sub  sp sp #1
    jr   rads
    ;; ---- Func 0 2
  :label_30
    add  bp bp #3
    st   @0 bp rads
    ;; ---- save regs
    st   @1 sp r1
    st   @2 sp r2
    add  sp sp #2
    ;; ---- Lval 2
    ;; ---- Lit 0
    ;; ---- Set
    ;; ---- 15:   i = 0;
    st   @-2 bp r0
    ;; ---- Rval 2
  :label_42
    ;; ---- Rvalg 1
    ;; ---- Lt
    ld   r1 @-2 bp
    ld   r2 2001
    sub  rz r1 r2
    jmp  ge label_118
    ;; ---- Jz 118
    ;; ---- 16:   while( i<N ){
    ;; ---- Lval 1
    ;; ---- Lit 0
    ;; ---- Set
    ;; ---- 17:     j = 0;
    st   @-1 bp r0
    ;; ---- Rval 1
  :label_59
    ;; ---- Rvalg 1
    ;; ---- Lt
    ld   r1 @-1 bp
    ld   r2 2001
    sub  rz r1 r2
    jmp  ge label_104
    ;; ---- Jz 104
    ;; ---- 18:     while( j<N ){
    ;; ---- Lvalg 2
    ;; ---- Rval 2
    ;; ---- Rval 1
    ;; ---- Call 208
    ld   r1 @-1 bp
    st   @2 bp r1
    ld   r1 @-2 bp
    st   @1 bp r1
    jal  rads label_208
    or   r1 retv rz
    ;; ---- Index
    ld   r2 #2002
    add  r1 r2 r1
    ;; ---- Rval 2
    ;; ---- Lit 1
    ;; ---- Add
    ld   r2 @-2 bp
    add  r2 r2 #1
    ;; ---- Set
    ;; ---- 19:       a[index(i,j)] = i+1;
    st   @0 r1 r2
    ;; ---- Lval 1
    ;; ---- Rval 1
    ;; ---- Lit 1
    ;; ---- Add
    ld   r1 @-1 bp
    add  r1 r1 #1
    ;; ---- Set
    ;; ---- 20:       j = j + 1;
    st   @-1 bp r1
    ;; ---- Jmp 59
    jmp always label_59
    ;; ---- Lval 2
  :label_104
    ;; ---- Rval 2
    ;; ---- Lit 1
    ;; ---- Add
    ld   r1 @-2 bp
    add  r1 r1 #1
    ;; ---- Set
    ;; ---- 22:     i = i + 1;
    st   @-2 bp r1
    ;; ---- Jmp 42
    jmp always label_42
    ;; ---- Ret0
  :label_118
    ld   rads @0 bp
    sub  bp bp #3
    ;; ---- restore regs
    ld   r1 @-1 sp
    ld   r2 @0 sp
    sub  sp sp #2
    jr   rads
    ;; ---- Func 0 2
  :label_119
    add  bp bp #3
    st   @0 bp rads
    ;; ---- save regs
    st   @1 sp r1
    st   @2 sp r2
    add  sp sp #2
    ;; ---- Lval 2
    ;; ---- Lit 0
    ;; ---- Set
    ;; ---- 27:   i = 0;
    st   @-2 bp r0
    ;; ---- Rval 2
  :label_131
    ;; ---- Rvalg 1
    ;; ---- Lt
    ld   r1 @-2 bp
    ld   r2 2001
    sub  rz r1 r2
    jmp  ge label_148
    ;; ---- Jz 148
    ;; ---- 28:   while( i<N ){
    ;; ---- Lval 1
    ;; ---- Lit 0
    ;; ---- Set
    ;; ---- 29:     j = 0;
    st   @-1 bp r0
    ;; ---- Rval 1
  :label_148
    ;; ---- Rvalg 1
    ;; ---- Lt
    ld   r1 @-1 bp
    ld   r2 2001
    sub  rz r1 r2
    jmp  ge label_193
    ;; ---- Jz 193
    ;; ---- 30:     while( j<N ){
    ;; ---- Lvalg 18
    ;; ---- Rval 2
    ;; ---- Rval 1
    ;; ---- Call 208
    ld   r1 @-1 bp
    st   @2 bp r1
    ld   r1 @-2 bp
    st   @1 bp r1
    jal  rads label_208
    or   r1 retv rz
    ;; ---- Index
    ld   r2 #2018
    add  r1 r2 r1
    ;; ---- Rval 1
    ;; ---- Lit 1
    ;; ---- Add
    ld   r2 @-1 bp
    add  r2 r2 #1
    ;; ---- Set
    ;; ---- 31:       b[index(i,j)] = j+1;
    st   @0 r1 r2
    ;; ---- Lval 1
    ;; ---- Rval 1
    ;; ---- Lit 1
    ;; ---- Add
    ld   r1 @-1 bp
    add  r1 r1 #1
    ;; ---- Set
    ;; ---- 32:       j = j + 1;
    st   @-1 bp r1
    ;; ---- Jmp 148
    jmp always label_148
    ;; ---- Lval 2
  :label_193
    ;; ---- Rval 2
    ;; ---- Lit 1
    ;; ---- Add
    ld   r1 @-2 bp
    add  r1 r1 #1
    ;; ---- Set
    ;; ---- 34:     i = i + 1;
    st   @-2 bp r1
    ;; ---- Jmp 131
    jmp always label_131
    ;; ---- Ret0
  :label_207
    ld   rads @0 bp
```

```
  sub  bp bp #3
  ;; ---- restore regs
  ld   r1 @-1 sp
  ld   r2 @0 sp
  sub  sp sp #2
  jr   rads
  ;; ---- Func 2 0
:label_208
  add  bp bp #3
  ;; ---- save regs
  st   @1 sp r1
  st   @2 sp r2
  add  sp sp #2
  ;; ---- Rval 2
  ;; ---- Rvalg 1
  ;; ---- Mul
  ld   r1 @-2 bp
  ld   r2 2001
  mul  r1 r1 r2
  ;; ---- Rval 1
  ;; ---- Add
  ld   r2 @-1 bp
  add  r1 r1 r2
  ;; ---- Ret1
  ;; ---- 39:   return i*N + j;
  or   retv r1 rz
  sub  bp bp #3
  ;; ---- restore regs
  ld   r1 @-1 sp
  ld   r2 @0 sp
  sub  sp sp #2
  jr   rads
  ;; ---- Func 0 4
:label_225
  add  bp bp #5
  st   @0 bp rads
  ;; ---- save regs
  st   @1 sp r1
  st   @2 sp r2
  st   @3 sp r3
  add  sp sp #3
  ;; ---- Lval 4
  ;; ---- Lit 0
  ;; ---- Set
  ;; ---- 43:   i = 0;
  st   @-4 bp r0
  ;; ---- Rval 4
:label_237
  ;; ---- Rvalg 1
  ;; ---- Lt
  ld   r1 @-4 bp
  ld   r2 2001
  sub  rz r1 r2
  jmp  ge label_384
  ;; ---- Jz 384
  ;; ---- 44:   while( i<N ){
  ;; ---- Lval 3
  ;; ---- Lit 0
  ;; ---- Set
  ;; ---- 45:     j = 0;
  st   @-3 bp r0
  ;; ---- Rval 3
:label_254
  ;; ---- Rvalg 1
  ;; ---- Lt
  ld   r1 @-3 bp
  ld   r2 2001
  sub  rz r1 r2
  jmp  ge label_370
  ;; ---- Jz 370
  ;; ---- 46:     while( j<N ){
  ;; ---- Lval 2
  ;; ---- Lit 0
  ;; ---- Set
  ;; ---- 47:       s = 0;
  st   @-2 bp r0
  ;; ---- Lval 1
  ;; ---- Lit 0
  ;; ---- Set
  ;; ---- 48:       k = 0;
  st   @-1 bp r0
  ;; ---- Rval 1
:label_278
  ;; ---- Rvalg 1
  ;; ---- Lt
  ld   r1 @-1 bp
  ld   r2 2001
  sub  rz r1 r2
  jmp  ge label_339
  ;; ---- Jz 339
  ;; ---- 49:       while( k<N ){
  ;; ---- Lval 2
  ;; ---- Rval 2
  ;; ---- Lvalg 2
  ;; ---- Rval 4
  ;; ---- Rval 1
  ;; ---- Call 208
  ld   r1 @-1 bp
  st   @2 bp r1
  ld   r1 @-4 bp
  st   @1 bp r1
  jal  rads label_208
  or   r1 retv rz
  ;; ---- Index
  ld   r2 #2002
  add  r1 r2 r1
  ;; ---- Fetch
  ld   r1 @0 r1
  ;; ---- Lvalg 18
  ;; ---- Rval 1
  ;; ---- Rval 3
  ;; ---- Call 208
  ld   r2 @-3 bp
  st   @2 bp r2
  ld   r2 @-1 bp
  st   @1 bp r2
  jal  rads label_208
  or   r2 retv rz
  ;; ---- Index
  ld   r3 #2018
  add  r2 r3 r2
  ;; ---- Fetch
  ld   r2 @0 r2
  ;; ---- Mul
  mul  r1 r1 r2
  ;; ---- Add
  ld   r2 @-2 bp
  add  r1 r2 r1
  ;; ---- Set
  ;; ---- 50:         s = s + a[index(i,k)] * b
[index(k,j)];
  st   @-2 bp r1
  ;; ---- Lval 1
  ;; ---- Rval 1
  ;; ---- Lit 1
  ;; ---- Add
  ld   r1 @-1 bp
  add  r1 r1 #1
  ;; ---- Set
  ;; ---- 51:         k = k + 1;
  st   @-1 bp r1
  ;; ---- Jmp 278
  jmp always label_278
  ;; ---- Lvalg 34
:label_339
  ;; ---- Rval 4
  ;; ---- Rval 3
  ;; ---- Call 208
  ld   r1 @-3 bp
  st   @2 bp r1
  ld   r1 @-4 bp
  st   @1 bp r1
  jal  rads label_208
  or   r1 retv rz
  ;; ---- Index
  ld   r2 #2034
  add  r1 r2 r1
  ;; ---- Rval 2
  ;; ---- Set
  ;; ---- 53:       c[index(i,j)] = s;
  ld   r2 @-2 bp
  st   @0 r1 r2
  ;; ---- Lval 3
  ;; ---- Rval 3
  ;; ---- Lit 1
  ;; ---- Add
  ld   r1 @-3 bp
  add  r1 r1 #1
  ;; ---- Set
  ;; ---- 54:       j = j + 1;
  st   @-3 bp r1
  ;; ---- Jmp 254
  jmp always label_254
  ;; ---- Lval 4
:label_370
  ;; ---- Rval 4
  ;; ---- Lit 1
  ;; ---- Add
  ld   r1 @-4 bp
  add  r1 r1 #1
  ;; ---- Set
  ;; ---- 56:     i = i + 1;
  st   @-4 bp r1
  ;; ---- Jmp 237
  jmp always label_237
  ;; ---- Ret0
:label_384
  ld   rads @0 bp
  sub  bp bp #5
  ;; ---- restore regs
  ld   r1 @-2 sp
  ld   r2 @-1 sp
  ld   r3 @0 sp
  sub  sp sp #3
  jr   rads
  ;; ---- Func 0 2
:label_385
  add  bp bp #3
  st   @0 bp rads
  ;; ---- save regs
  st   @1 sp r1
  st   @2 sp r2
  add  sp sp #2
  ;; ---- Lval 2
  ;; ---- Lit 0
  ;; ---- Set
  ;; ---- 61:   i = 0;
  st   @-2 bp r0
  ;; ---- Rval 2
:label_397
```

```
   ;; ---- Rvalg 1
   ;; ---- Lt
   ld   r1 @-2 bp
   ld   r2 2001
   sub  rz r1 r2
   jmp  ge label_479
   ;; ---- Jz 479
   ;; ---- 62:   while( i<N ){
   ;; ---- Lval 1
   ;; ---- Lit 0
   ;; ---- Set
   ;; ---- 63:     j = 0;
   st   @-1 bp r0
   ;; ---- Rval 1
:label_414
   ;; ---- Rvalg 1
   ;; ---- Lt
   ld   r1 @-1 bp
   ld   r2 2001
   sub  rz r1 r2
   jmp  ge label_457
   ;; ---- Jz 457
   ;; ---- 64:     while( j<N ){
   ;; ---- Lvalg 34
   ;; ---- Rval 2
   ;; ---- Rval 1
   ;; ---- Call 208
   ld   r1 @-1 bp
   st   @2 bp r1
   ld   r1 @-2 bp
   st   @1 bp r1
   jal  rads label_208
   or   r1 retv rz
   ;; ---- Index
   ld   r2 #2034
   add  r1 r2 r1
   ;; ---- Fetch
   ld   r1 @0 r1
   ;; ---- Print
   trap print r1
   ;; ---- Lit 32
   ;; ---- Printch
   ld   r1 #32
   trap printch r1
   ;; ---- Lval 1
   ;; ---- Rval 1
   ;; ---- Lit 1
   ;; ---- Add
   ld   r1 @-1 bp
   add  r1 r1 #1
   ;; ---- Set
   ;; ---- 66:       j = j + 1;
   st   @-1 bp r1
   ;; ---- Jmp 414
   jmp always label_414
   ;; ---- Lit 13
:label_457
   ;; ---- Printch
   ld   r1 #13
   trap printch r1
   ;; ---- Lit 10
   ;; ---- Printch
   ld   r1 #10
   trap printch r1
   ;; ---- Lval 2
   ;; ---- Rval 2
   ;; ---- Lit 1
   ;; ---- Add
   ld   r1 @-2 bp
   add  r1 r1 #1
   ;; ---- Set
   ;; ---- 69:     i = i + 1;
   st   @-2 bp r1
   ;; ---- Jmp 397
   jmp always label_397
   ;; ---- Ret0
:label_479
   ld   rads @0 bp
   sub  bp bp #3
   ;; ---- restore regs
   ld   r1 @-1 sp
   ld   r2 @0 sp
   sub  sp sp #2
   jr   rads

.e
====================

   1 Call 5
   4 Stop
   5 Func 0 0
  10 Lvalg 1
  13 Lit 4
  16 Set
  17 Call 30
  20 Call 119
  23 Call 225
  26 Call 385
  29 Ret0
  30 Func 0 2
  35 Lval 2
  38 Lit 0
  41 Set
  42 Rval 2
  45 Rvalg 1
  48 Lt
  49 Jz 118
  52 Lval 1
  55 Lit 0
  58 Set
  59 Rval 1
  62 Rvalg 1
  65 Lt
  66 Jz 104
  69 Lvalg 2
  72 Rval 2
  75 Rval 1
  78 Call 208
  81 Index
  82 Rval 2
  85 Lit 1
  88 Add
  89 Set
  90 Lval 1
  93 Rval 1
  96 Lit 1
  99 Add
 100 Set
 101 Jmp 59
 104 Lval 2
 107 Rval 2
 110 Lit 1
 113 Add
 114 Set
 115 Jmp 42
 118 Ret0
 119 Func 0 2
 124 Lval 2
 127 Lit 0
 130 Set
 131 Rval 2
 134 Rvalg 1
 137 Lt
 138 Jz 207
 141 Lval 1
 144 Lit 0
 147 Set
 148 Rval 1
 151 Rvalg 1
 154 Lt
 155 Jz 193
 158 Lvalg 18
 161 Rval 2
 164 Rval 1
 167 Call 208
 170 Index
 171 Rval 1
 174 Lit 1
 177 Add
 178 Set
 179 Lval 1
 182 Rval 1
 185 Lit 1
 188 Add
 189 Set
 190 Jmp 148
 193 Lval 2
 196 Rval 2
 199 Lit 1
 202 Add
 203 Set
 204 Jmp 131
 207 Ret0
 208 Func 2 0
 213 Rval 2
 216 Rvalg 1
 219 Mul
 220 Rval 1
 223 Add
 224 Ret1
 225 Func 0 4
 230 Lval 4
 233 Lit 0
 236 Set
 237 Rval 4
 240 Rvalg 1
 243 Lt
 244 Jz 384
 247 Lval 3
 250 Lit 0
 253 Set
 254 Rval 3
 257 Rvalg 1
 260 Lt
 261 Jz 370
 264 Lval 2
 267 Lit 0
 270 Set
 271 Lval 1
 274 Lit 0
 277 Set
 278 Rval 1
 281 Rvalg 1
 284 Lt
 285 Jz 339
 288 Lval 2
 291 Rval 2
 294 Lvalg 2
 297 Rval 4
 300 Rval 1
```

```
303 Call 208
306 Index
307 Fetch
308 Lvalg 18
311 Rval 1
314 Rval 3
317 Call 208
320 Index
321 Fetch
322 Mul
323 Add
324 Set
325 Lval 1
328 Rval 1
331 Lit 1
334 Add
335 Set
336 Jmp 278
339 Lvalg 34
342 Rval 4
345 Rval 3
348 Call 208
351 Index
352 Rval 2
355 Set
356 Lval 3
359 Rval 3
362 Lit 1
365 Add
366 Set
367 Jmp 254
370 Lval 4
373 Rval 4
376 Lit 1
379 Add
380 Set
381 Jmp 237
384 Ret0
385 Func 0 2
390 Lval 2
393 Lit 0
396 Set
397 Rval 2
400 Rvalg 1
403 Lt
404 Jz 479
407 Lval 1
410 Lit 0
413 Set
414 Rval 1
417 Rvalg 1
420 Lt
421 Jz 457
424 Lvalg 34
427 Rval 2
430 Rval 1
433 Call 208
436 Index
437 Fetch
438 Print
439 Lit 32
442 Printch
443 Lval 1
446 Rval 1
449 Lit 1
452 Add
453 Set
454 Jmp 414
457 Lit 13
460 Printch
461 Lit 10
464 Printch
465 Lval 2
468 Rval 2
471 Lit 1
474 Add
475 Set
476 Jmp 397
479 Ret0
```