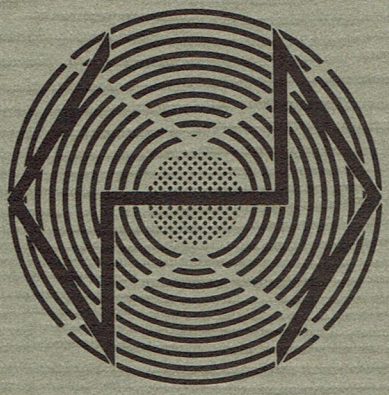
FVN



A/S REGNECENTRALEN

SCANDINAVIAN INFORMATION PROCESSING SYSTEMS

RC SYSTEM LIBRARY: FALKONERALLE 1 · DK-2000 COPENHAGEN F

RCSL No: 31-D280

Edition: July 1973

Author: Torkild Glaven,
minor changes: Jens Ramsbøl

Title: DO

Keywords: RC 4000, Basic Software, File Processor Utility Program

Abstract: The do language is intended for monitor and hardware testing. However it is designed so that it can be used as a general programming language. 21 pages.

FILE PROCESSOR UTILITY PROGRAM: DO

ABSTRACT:

The do language is intended for monitor and hardware testing and supervising, but it is designed so that it can be used as a general programming language, in which the user can set register and core store contents, execute monitor and FP procedures as well as slang instructions, output any information about the core store in an easy readable form etc.

Examples:

The command:

```
do write 102.word.4 116
```

will output the contents of words 102, 104 and 116, i.e. the values of max time slice, time slice and number of storage bytes.

The command:

```
do w1.w2  w2.w3.in.20  140.w3
```

will execute w1:= w2, w2:= w3 + 20, word 140:= w3.

The command:

```
outfile=do w0.74,                    ; w0:= 74
           w0.x0.0,                  ; w0:= word(w0)
           w1.76,                    ; w1:= 76
           z1.x1.0,                  ; z1:= word(w1)
           do w0.in.2,               ; w0:= w0 + 2
              while w0.ne.z1,        ; while w0 <> z1
              w2.x0.0,               ; w2:= word(w0)
              write x2.peripheral.26 end,
           ,                         ; write 26 bytes of x2 with
           ,                         ; layout peripheral
           od                        ; end inner do
```

will on outfile output the first 26 bytes of the descriptions of all peripheral devices.

See also further examples sec. 6

1. SYNTAX:

The program is called as follows:

<output file> = do <s> <do command>

<do command> ::= { <assignment> | <write command> | <monitor call> | <fp call> | <slang command> | jump <s> <value> | clear | <wait command> | if <s> <value> | fi | do | while <s> <value> | od | go <s> <value> | og | exit }

<wait command> ::= wait <s> { <integer> | <name> }

<value> ::= <expression> { .<expression> }$_0^\infty$

<assignment> ::= { <simple variable> <expression tail> | <array base> } { .<name> | .<expression> }$_0^\infty$

<simple variable> ::= { w | z } { 0 | 1 | 2 | 3 }

<array base> ::= { <integer> | <array name> }

<array name> ::= { x / y } { 0 / 1 / 2 / 3 }

<expression tail> ::= { . <operator> . <operand> }$_{0}^{\infty}$

<expression> ::= <operand> <expression tail>

<operand> ::= { <integer> / <simple variable> / <indexed variable> / ba / cc / fp / hn }

<indexed variable> ::= <array name> . <integer>

<operator> ::=	
in	+
de	-
le	shift left
ri	shift right
or	logical or
an	logical and
eq	=
ne	<>
gr	>
ls	<
ng	<=
nl	>=
mu	*
di	//
mo	modulo
po	**

<write command> ::= write <s> <write action> { end / <end of do call> }

<write action> ::= { <simple variable> { . <format> }$_0^\infty$ | <base> { . <bytes> | . <format> }$_0^\infty$ }

<base> ::= { <array base> | ba | cc | fp | hn | bittable }

<bytes> ::= { <integer> | <simple variable> | <indexed variable> }

<format> ::= { <simple format> | <structured format> }

<simple format> ::= { empty | word | bytes | octets | sixtets | octal | binary | text | code | double | groups | all | binword | words5 | bytes10 | name | procname | procnames }

empty	nothing is output
word	word as signed integer
bytes	2 bytes
octets	3 8-bit groups
sixtets	4 6-bit groups
octal	positive octal number
binary	bit pattern
text	3 iso-characters
code	slang instruction
double	2 words as double word
groups	word,bytes,octets,sixtets,octal
all	code,octal,word,bytes,octets,text
binword	word and binary
words5	5 words/line
bytes10	10 bytes/line
name	4 words as text
procname	
procnames	

$$\langle\text{structured format}\rangle ::= \left\{\begin{array}{l}\text{buffer}\\ \text{area}\\ \text{peripheral}\\ \text{internal}\\ \text{tail}\\ \text{zone}\\ \text{share}\\ \langle\text{mon. 2 format}\rangle\\ \langle\text{mon. 3 format}\rangle\end{array}\right\}$$

$\langle\text{mon. 2 format}\rangle ::= \text{note}$

$$\langle\text{mon. 3 format}\rangle ::= \left\{\begin{array}{l}\text{answer}\\ \text{chaintable}\\ \text{entry}\end{array}\right\}$$

$$\langle\text{slang command}\rangle ::= \text{slang } \langle\text{s}\rangle\ \langle\text{value}\rangle \left\{\begin{array}{l}\langle\text{s}\rangle\ \langle\text{instruction}\rangle\\ \langle\text{s}\rangle\ \text{w}\ \langle\text{word}\rangle\end{array}\right\}_{0}^{\infty} \left\{\begin{array}{l}\text{end}\\ \langle\text{end of do call}\rangle\end{array}\right\}$$

$$\langle\text{instruction}\rangle ::= \langle\text{code}\rangle \left\{\begin{array}{l}\left\{\begin{array}{l}\langle\text{s}\rangle\\ .\end{array}\right\}\left\{\begin{array}{l}\text{w}\\ \text{x}\end{array}\right\}\left\{\begin{array}{l}0\\ 1\\ 2\\ 3\end{array}\right\}\\ \langle\text{word}\rangle\end{array}\right\}_{0}^{\infty}$$

$$\langle\text{word}\rangle ::= \left\{\begin{array}{l}\left\{\begin{array}{l}\langle\text{s}\rangle\\ .\end{array}\right\}\left\{\begin{array}{l}\left\{\begin{array}{l}\text{y}\\ \text{z}\end{array}\right\}\left\{\begin{array}{l}0\\ 1\\ 2\\ 3\end{array}\right\}\\ \langle\text{integer}\rangle\end{array}\right\}\end{array}\right\}_{0}^{\infty}$$

$\langle\text{code}\rangle ::= \langle\text{two letter mnemonic slang instruction}\rangle$

<monitor call> ::=
- monitor <s> procedure <s> <integer>
- set <s> interrupt
- process <s> description
- initialize <s> process
- reserve <s> process
- release <s> process
- include <s> user
- exclude <s> user
- send <s> message
- wait <s> answer
- wait <s> message
- send <s> answer
- wait <s> event
- get <s> event
- get <s> clock
- set <s> clock
- create <s> entry
- look <s> up <s> entry
- change <s> entry
- rename <s> entry
- remove <s> entry
- permanent <s> entry
- create <s> area <s> process
- create <s> peripheral <s> process
- create <s> internal <s> process
- start <s> internal <s> process
- stop <s> internal <s> process
- modify <s> internal <s> process
- remove <s> process
- generate <s> name
- copy
- <mon. 2 call>
- <mon. 3 call>

<mon. 2 call> ::= { modify <s> backing <s> store
| select <s> backing <s> store
| select <s> mask
| test <s> log
| return <s> status }

<mon. 3 call> ::= { set <s> catalog <s> base
| set <s> entry <s> base
| lookup <s> head <s> and <s> tail
| set <s> backing <s> storage <s> claims
| create <s> pseudo <s> process
| regret <s> message
| create <s> backing <s> storage
| insert <s> entry
| remove <s> backing <s> storage
| permanent <s> entry <s> in <s>
  auxiliary <s> catalog
| create <s> entry <s> look <s> process }

<fp call> ::=

fp <s> procedure <s> <integer>
finis <s> message
inblock <s> current
inblock
outblock <s> current
outblock
wait <s> ready <s> input
wait <s> ready <s> output
wait <s> ready
inchar <s> current
inchar
outchar <s> current
outchar
connect <s> current <s> input
connect <s> input
connect <s> current <s> output
connect <s> output
stack <s> input
stack <s> zone
unstack <s> input
unstack <s> zone
outtext <s> current
outtext
outinteger <s> current
outinteger
outend <s> current
outend
closeup <s> current
closeup
parent <s> message
wait <s> free <s> input
wait <s> free <s> output
wait <s> free
break <s> message
terminate <s> input
terminate <s> output
terminate <s> zone

2. FUNCTION:

The program processes the list of commands given in the program call. All variables are placed so that they remain unchanged by successive calls. A sequence of commands can therefore be executed in one or several do calls, provided these calls are of same kind (all with or without specified output file).

(numbers in [] refer to the examples in section 5)

w0, w1, w2, w3, z0, z1, z2, and z3 are names of simple variables [1,3].

x0, x1, x2, x3, y0, y1, y2, and y3 are names of arrays. An array name followed by a point and an integer acts as an indexed variable [2,3,5,7]. The start address of an array depends on the context.

An expression is a list of alternating operands and operators separated by points [1,2]. The list must begin and end with an operand. An expression is interpreted from left to right. The result is an integer word. The operands and their values are:

<integer>	the integer value.
w0-w3,z0-z3	the value of the variable.
x0-x3, indexed	the word addressed by the value of the corresponding w-variable increased by the index.
y0-y3, indexed	the word addressed by the value of the corresponding z-variable increased by the index.
ba	the buffer address (first free byte).
cc	the current command address (the byte after the last free byte).
fp	the fp base.
hn	the address of a word array containing the fp h-names (h0-h99).

All operators are dyadic and have the same priority [24]. The notation, the meaning, and the effect of the operators are (left and right refer to the operands):

in	increase	left + right
de	decrease	left - right
le	left shift (logical)	left shift right
ri	right shift (logical)	left shift (-right)

or	or (logical)	left or right
an	and (logical)	left and right
eq	equal	if left=right then -1 else 0
ne	not equal	if left<>right then -1 else 0
gr	greater	if left>right then -1 else 0
ls	less	if left<right then -1 else 0
ng	nog greater	if left<=right then -1 else 0
nl	not less	if left>=right then -1 else 0
mu	multiply	left * right
di	divide	left / right
mo	modulo	left mod right
po	power	left ** right

2.1. ASSIGNMENTS TO SIMPLE VARIABLES:

An expression where first operand is a name of a simple variable assigns the result to the variable [38]. If the expression is followed by one or more other expressions or names, the result of the last expression or the first word of the last name is assigned to the variable [2,4].

2.2. ASSIGNMENTS TO ARRAYS:

An array assignment consists of a start address (an integer or an array name) followed by a list of expressions and names stored in consecutive words. Expression values are stored as integer words, names as four text words [5,7].

Four fields corresponding to the names x0-x3 are reserved for building up arrays. Each field has a size of 34 bytes. An assignment to one of the names x0-x3 will place the array in the corresponding field, and transfer the address of this field to the w-variable of same number.

When assigning to y-names the value of the corresponding w-variable will be used as start address. (About the use of x- and y-arrays in expressions, see p. 9, in write actions, p. 11, and in slang, p. 13).

An integer used as start address is interpreted as an absolute address in core.

2.3. WRITE COMMAND:

Output from the do-program is controlled by write commands. Current output is used unless the call specifies another file, in which case current program zone (h19) is used.

Notation: write <write actions separated by spaces> end
The terminating word (end) may be omitted at the end of the parameter list.
All output concerns core store content. There are three kinds of write actions:

1. simple variable write action
2. array write action
3. special write action

1. simple variable write action:
The write action consists of the name of the simple variable [9]. Initially the format is integer word. This can be changed by one or more formats (see later) [10].

2. array write action:
The write action consists of a write base defining the start address [13]. The write base can be an integer using this as the start address, an x-name using the value of the corresponding w-variable as start address, an y-name using the value of the corresponding z-variable as start address, or one of the constant operands, ba, cc, fp and hn, using this value as start address. Initially the number of bytes to be output is two and the format is integer word. Bytes and format can be changed by placing parameters after the write base [13]. Bytes can be defined by an integer, a simple variable, or an indexed variable. For each definition the number of bytes will be output using the current format. The format can be changed by one or more formats (see later). Simple formats define how words or small groups of words should be output. Each word or group of words is output in the same way [15]. A structured format defines a relative start address and a structure of several simple formats. Further it defines the number of bytes to be output [37], unless this is specified after the format.

3. special write action:
This acts as an array write action except for the write base which is a name defining a special set of start address, initial number of bytes, and initial format [16].

The special write actions are:

bittable — the bittable placed at the top of the core store contains one bit for each backing store segment (only used in monitor 2).
Format: binary.

The simple formats are:

empty — nothing is output
word — signed integer
bytes — two bytes as positive integers
octets — three 8-bit groups as positive integers
sixtets — four 6-bit groups as positive integers
octal — a positive octal number
binary — binary number with points instead of zeroes
text — three 8-bit groups as iso characters. Character values less than 32 are output as spaces.
code — slang instructions
double — two words as a double word integer
groups — a combination of word, bytes, octets, sixtets, and octal
binword — a combination of word and binary
all — a combination of code, octal, word, bytes, octets, and text
words5 — five words on one line
bytes10 — ten bytes on one line
name — four words as a text
procname — a signed integer followed by a point and the name of the process having the word value as process description address
procnames — a binary word followed by the names of the internal processes, the identification bits of which match the ones in the word value.

The structured formats are:
(the words are output using simple formats corresponding to their contents). In the paranthesis is specified the standardlength in bytes for monitor 2 and monitor 3.

buffer — a message buffer (24, 24)
area — an area process description (20, 24)
peripheral — a peripheral process description (90, 100)
internal — an internal process description (74, 92)
tail — a catalog entry tail (20, 20)

zone	a zone descriptor (50, 50)
share	a share descriptor (24, 24)
note	an fp note (22, -)
answer	an answer from external process (10, 10)
entry	an area entry (-, 36)
chaintable	a chaintable head and a part of the table (-, 66)

Only the last format of a sequence of formats is used. For all write actions output takes place only after an explicit definition of bytes or at the end of the write action.

2.4. CALL OF MONITOR PROCEDURES:

A monitor procedure will be called when its name appears in the parameter list [35]. The call uses the variables w0-w3 as register values. On return the registers are stored in these variables. The names of the procedures can be found in the syntax description. New procedures will be included when appearing. The first procedure has the following effect:

monitor <s> procedure <s> <integer> jd 1<11+<integer>

2.5. CALL OF FILE PROCESSOR PROCEDURES:

A file processor procedure will be called when its name appears in the parameter list [18]. The call uses the variables w0-w3 as register values. On return the registers are stored in these variables. The names of the procedures can be found in the syntax description. New procedures will be included when appearing. The first procedure has the following effect:

fp <s> procedure <s> <integer> jl w3 <fpbase>+<integer>

2.6. SLANG COMMAND:

Notation: slang <value> <instructions> end

Function: stores a list of instructions and word values in consecutive words starting with the address defined by <value> [50]. An instruction starts with a mnemonic code. Modifications are determined thus:

<s> w-name	working register
. w-name	relative mark and working register
<s> x-name	index register
. x-name	indirect mark and index register.

Only the last modification of each kind is used. A word value starts with the letter, w. The word value and the instruction displacement are determined as the sum of a zero and possible appearings of the values:

<s> integer + <integer>
. integer - <integer>
<s> y-name + <value of w-variable>
. y-name - <value of w-variable>
<s> z-name + <value of z-variable>
. z-name - <value of z-variable>

2.7. JUMP COMMAND:

Notation: jump <value>

Function: jumps to the address defined by <value> with link in w3. The variables w0, w1, and w2 are used as register values. On return w0, w1, and w2 are stored in these variables [58].

2.8. CLEAR COMMAND:

Notation: clear

Function: clears all variables and the buffer area by setting zeroes into the process area not occupied by the file processor and the do-program itself.

2.9. WAIT COMMAND:

Notation: wait <s> <integer>
or wait <s> <name>

Function: An integer denotes the number of seconds (CPU-time) in which dummy instructions are executed. A name is assumed to be a process name and the do-program tries to reserve the process until it succeeds doing this. If the name is not a process name it is looked up in the catalog and its possible document name is used as process name [8].

2.10. IF COMMAND:

Notation: if <s> <expression>

Function: if the expression is negative (true), the command has no effect. If it is positive (false), commands up to and including the corresponding fi command are skipped. This means that nesting of conditions is possible [26].

2.11. FI COMMAND:

Notation: fi

Function: No effect [26], but see the if command.

2.12. DO COMMAND:

Notation: do

Function: Stacks a return point for the corresponding od command [39].

2.13. WHILE COMMAND:

Notation: while <s> <expression>

Function: If the expression is negative (true), the command has no effect. If it is positive (false), commands up to and including the corresponding od command are skipped and the command pointer is unstacked. Unstacking the outermost command pointer has no effect [28].

2.14. OD COMMAND:

Notation: od

Function: Interpretation continues at the stacked command pointer [29], see do command.

2.15. GO COMMAND:

Notation: go <value>

Function: The command is a procedure declaration head or a procedure call. <value> defines the procedure number which must not be negative or above a certain limit which for the moment is 30. The first appearance of a command with a given number acts as the head of a procedure declaration [43]. The command pointer is saved and the procedure body terminated by the corresponding og-command is skipped. Later appearances of a go-command act as procedure calls [48]. The command pointer is stacked and interpretation continues at the saved procedure command pointer.

2.16. OG COMMAND:

Notation: og

Function: The command pointer is unstacked and interpretation continues at this command pointer [46].

2.17. EXIT COMMAND:

Notation: exit

Function: Termination of the program [33].

3. STORAGE REQUIREMENTS:

5730 bytes plus the FILE PROCESSOR.

4. MESSAGES:

Appearing on current output.

***do param <illegal parameter>

parameter in illegal syntactical position. The parameter is ignored.

***do connect <i> <outfile>

<outfile> could not be connected for output because of a hard error. The value of <i> determines the error:

1 no resources
2 malfunctioning
3 not user, non-exist
4 convention error
5 not allowed
6 name format error

The ok-bit is set to false and the program is terminated.

***do no core

the core area is too small. The ok bit is set to false and the program is terminated.

***do core addr

attempt to use a storage word outside the core store. The ok bit is set to false and the program is terminated.

***do format

error in the format table (error in the do-program). The ok bit is set to false and the program is terminated.

***do niveau

the number of format niveaus is exceeded. The number is an assembly option. Indicates possibly an error in the format table (error in the do-program). The ok bit is set to false and the program is terminated.

5. EXAMPLES:

(numbers in [] used for references in section 2)

```
[1]   do w0. 17                    ; w0:= 17;
[2]   do w1.cc.x1.0                ; w1:= current command;
                                   ; w1:= word(w1);
[3]   do z2.cc.y2.2                ; z2:= current command;
                                   ; z2:= word(z2+2);
[4]   do z3.longname               ; z3:= <:lon:>;
[5]   do x2.0.de.1.test.1.2        ; w2:= address of w2-field;
                                   ; word(w2):= 0-1;
                                   ; words(w2+2:w2+8):= <:test:>;
                                   ; word(q2+10):= 1;
                                   ; word(w2+12):= 2;
[6]   do w3.w2.in.10               ; w3:= w2+10;
[7]   do y3.w0                     ; word(w3):= w0;
[8]   do wait lp                   ; wait until lp is ready;
[9]   do write w0,                 ; write(out,w0,
[10]     w1.bytes,                 ;     <<bytes>,w1,
[11]     z2.octets,                ;     <<octets>,z2,
[12]     z3.text,                  ;     <<text>,z3,
[13]     x2.code.2,                ;     <<code>,word(w2),
[14]       .name.8,                ;     <<name>,words(w2+2:w2+8),
[15]       .word.4,                ;     <<word>,words(w2+10:w2+12),
[16]     bittable.2                ;     <<binary>,word(bittable));
```

The following example reads characters from current input and determines the two types: digit and other.

```
[17]  do,                          ; begin
[18]    inchar current,            ; start:
[19]    z0.w2 w2.10,               ;   inchar(i);
[20]    outchar current,           ;   outchar(10);
[21]    w2.z0 outchar,             ;   outchar(i);
[22]    w2.32 outchar,             ;   outchar(32);
[23]    z1.z0.ng.57,               ;   digit:= i<=57;
[24]    z2.z0.nl.48.an.z1,         ;   digit:= digit and i>=48;
[25]    x0.other,                  ;   text:= <:other:>;
[26]    if z2 x0.digit fi,         ;   if digit then text:= <:digit:>;
[27]    outtext,                   ;   outtext(text);
```

```
[28]    while z0.ls.124,          ;   if i<124 then
[29]  od,                         ;     goto start;
[30]  w2.10 outend,               ;   outend(10);
[31]  w1.hn.x1.40.in.fp,          ;   in:= fpbase+h20;
[32]  write x1.zone               ;   write(<<zone>,in)
      a1p3ø                       ; end a1p3ø

[33]  do exit so this is not executed
```

The following example prints the process description and the event queue for the internal process, s.

```
[34]  do x3.s,                    ; begin
[35]    process description,      ;   process description(<:s:>,proc);
[36]    if w0.gr.0,               ;   if proc>0 then
[37]      write x0.internal end,  ;     begin write(<<internal>,proc);
[38]      w0.in.14 w2.w0,         ;       head:= buf:= proc+14;
[39]      do w2.x2.0,             ;       for buf:= word(buf)
[40]        while w2.ne.w0,       ;           while buf<>head do
[41]        write x2.buffer end,  ;         write(<<buffer>,buf)
[42]      od                      ; end end;

[43]  do go 0,                    ; procedure go 0;
[44]      w0.in.1,                ; begin w0:= w0+1;
[45]      write w0 end,           ;   write(w0)
[46]    og,                       ; end;
[47]    clear,                    ; clear;
[48]    go 0 go 0                 ; go 0; go 0;
```

The following example assembles, prints, and executes a piece of code.

```
[49]  do w2.fp,                   ; w2:= fp base;
[50]    slang ba,                 ; slang(ba);
[51]      rs.w3    0,             ; begin save link;
[52]      al w0    11,            ;   w0:= 11;
[53]      al.w1    .4,            ;   w1:= ba;
[54]      ac w2 x2 0,             ;   w2:= -w2;
[55]      jl.w0.x0.8,             ;   goto saved link;
[56]    end,                      ; end;
[57]    write ba.code.10 end,     ; write(<<code>,words(ba:ba+8));
[58]    jump ba,                  ; jump(ba);
[59]    write ba w0 w1 w2 fp.0    ; write(ba,word(ba),w0,w1,w2,fp);
```

Output from examples:

```
w0 =        17

w1 =      2    10

z2 =  120 100 111

z3 =  lon

x2.234598
  +0  63.w3(x3-1)
  +2  test
 +10        17
 +12         2

bittable.262138
  +0  ......111111111111111.1.

a other
1 digit
p other
3 digit
ø other

x1.232350
 -36    244789
 -34    245301
 -32    233474
 -30    233474
 -28    233474

 -26  2048     0
 -24  boss
 -16     8900
 -14        0
 -12        0
 -10       69

  -8  ........................1
  -6   232034
  -4   10   1    0
  -2        0     0     0     0     0     0     0     0     0     0  00000000

  +0   244791
  +2   244793
  +4        0
  +6        0     0     0     0     0     0     0     0     0     0  00000000

  +8        0     0     0     0     0     0     0     0     0     0  00000000
 +10        0     0     0     0     0     0     0     0     0     0  00000000
 +12        0     0     0     0     0     0     0     0     0     0  00000000
```

```
x0.16804
  -4 -8388607
  -2  8388606
  +0        0
  +2  s
 +10  .........1......1...1111      4  143
 +12  .1.....................
 +14    16818
 +16    16818
 +18    16822
 +20    16822
 +22    32304
 +24   262144
 +26     3    1
 +28  ............111111111111      0 4095
 +30        0
 +32  ....1.....................    128    0
 +34  1..11111111111111111111
 +36    32304
 +38    37080      9  216    0 144 216    0  9  3 24  00110330
 +40    35992      8 3224    0 140 152    0  8 50 24  00106230
 +42        0      0    0    0   0   0    0  0  0  0  00000000
 +44        3      0    3    0   0   3    0  0  0  3  00000003
 +46        0
 +48    34090
 +50        0.
 +52      116
 +56                40448
 +60                  0
 +64     1767260780000
 +66    16818
 +68 -8388607
 +70  8388605
 +72 -8388607
 +74  8388605
 +76 -8388607
 +78  8388605
 +80     0    0
 +82     0    0
 +84     0    0
 +86     0    0

w0 =        1

w0 =        2

ba.239824
  +0  rs.w3 0      239824
  +2  al w0 11
  +4  al.w1 -4     239824
  +6  ac w2 x2+0
  +8  jl.  (-8)    239824

ba.239824
  +0   236310

w0 =       11

w1 =   239824

w2 =  -231990

fp.231990
```