



# N1 Manual

Dirk Heisswolf

April 26, 2019

---

## Revision History

Date	Change
March 21, 2019	Initial release
March 22, 2019	Changed encoding of ALU operands (see <a href="#">Table 2-1</a> and <a href="#">Table 3-1</a> )
April 17, 2019	Fixed some copy & paste errors
April 26, 2019	Fixes and a description of the branch condition in <a href="#">Section 2.4</a> “Conditional Branches”

## Contents

<b>1 Overview</b>	<b>5</b>
<b>2 Instruction Set</b>	<b>7</b>
2.1 Return from a Call (;)	8
2.2 Jump Instructions	8
2.3 Call Instructions	8
2.4 Conditional Branches	8
2.5 Literals	8
2.6 ALU Instructions	8
2.7 Stack Instructions	10
2.8 Memory Access Instructions	13
2.9 Control Instructions	13
<b>3 ANS Forth Words</b>	<b>15</b>
<b>4 Stacks</b>	<b>18</b>
4.1 Parameter Stack	18
4.2 Return Stack Stack	19
<b>5 Reset, Exceptions, and Interrupts</b>	<b>20</b>
5.1 Reset	20
5.2 Exceptions	20
5.3 Interrupts	21
<b>6 Integration Guide</b>	<b>22</b>
6.1 Integratation Parameters	22
6.2 Interfaces	22
6.3 Target Specific Design Files	26
<b>7 Verification Status</b>	<b>27</b>
<b>8 Tool Summary</b>	<b>28</b>
<b>9 Glossary</b>	<b>29</b>
<b>10 References</b>	<b>32</b>

**List of Figures**

2-1	Instruction encoding . . . . .	7
2-2	Transition encoding of stack instructions . . . . .	10
4-1	Stack Architecture . . . . .	18

## List of Tables

2-1	ALU operations . . . . .	9
2-2	Common stack operations . . . . .	11
2-2	Common stack operations . . . . .	12
2-2	Common stack operations . . . . .	13
2-3	Concurrent Control Instructions . . . . .	13
2-4	Non-concurrent control instruction encoding . . . . .	13
2-4	Non-concurrent control instruction encoding . . . . .	14
3-1	ANS Forth words . . . . .	15
3-1	ANS Forth words . . . . .	16
3-1	ANS Forth words . . . . .	17
5-1	Throw codes . . . . .	21
8-1	Tool Summary . . . . .	28

## 1 Overview

The N1 is a 16-bit stack machine, targeted for low-end FPGA applications. Its instruction set and architecture are designed for efficient execution of [Forth](#) code.

Here is a summary of the N1's characteristics:

### Memory connection:

- 16-bit [Von-Neumann-Architecture](#)
- Separate address space for [stack](#) content
- [Wishbone](#) interfaces to main and stack memory
- Up to 128KB (main) memory space
- Memory addressable in 16-bit entities only

### Stacks:

- Two hardware [stacks](#) ([parameter](#) and [return stack](#))
- Each [stack](#) consists of three segments:

#### [Upper stack:](#)

- Shift registers with selectable shift direction for each individual cell
- Fixed size
  - \* Upper [parameter stack](#): 4 [cells](#)
  - \* Upper [return stack](#): 1 [cell](#)

#### [Intermediate stack:](#)

- Buffer with lazy data transfers to and from the lower stack
- Configurable size

#### [Lower stack:](#)

- [RAM](#) space shared by both [stacks](#)
- [Stacks](#) grow towards each other
- Up to 128KB in size

### Instruction set:

- Fixed instruction size of 16-bit
- [Jumps](#) and [calls](#)
  - [Indirect addressing](#)
  - [Direct addressing](#) within a 32KB window
  - Two bus cycle execution time
  - Return from [calls](#) performed concurrently with last instruction
- [Conditional branches](#)
  - [Direct relative addressing](#) within a 16KB range
  - Two bus cycles of execution time if branch is taken, one cycle if not
- [Literals](#)
  - [Immediate](#) encoding of literals between -2048 and 2047
  - Literals out of this range require one additional instruction
- [Arithmetic and logic operations](#)
  - Single cycle [ALU](#) operations include:

- \* Sum and Difference
  - \* Comparisons
  - \* Signed and unsigned products
  - \* Bitwise logic operations
  - \* Multi-bit shifts
- Optional **immediate** encoding of one operand, using 5-bit encoding
- **Stack** operations
  - All 1024 stack transitions of the **upper stack** encodable
- Memory I/O
  - **Indirect addressing**
  - **Direct addressing** within a 511B window
  - Two bus cycle execution time if branch is taken, one cycle if not

**Exceptions:**

- Exception handler invoked by five error conditions:
  - **Parameter stack** overflow
  - **Parameter stack** underflow
  - **Return stack** overflow
  - **Return stack** underflow
  - Access violations in the (main) address space

**Interrupts:**

- Optional interrupt support through external interrupt controller
- Automatic interrupt acknowledge (flag clearing) supported

## 2 Instruction Set

The intent of the N1's instruction set is to map most of the essential Forth words to single cycle instructions. [Figure 2-1](#) illustrates the instruction format.

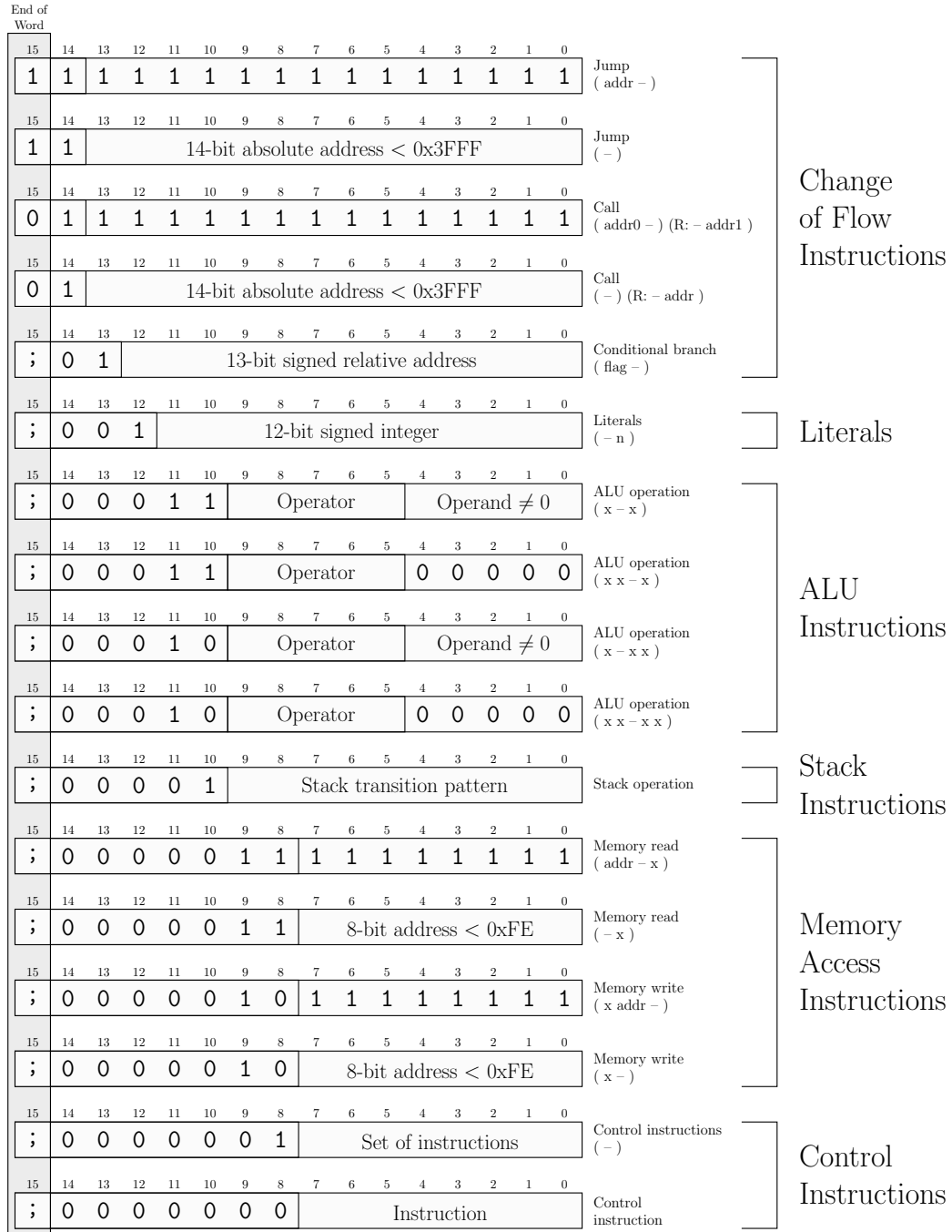


Figure 2-1: Instruction encoding



## 2.1 Return from a Call (;)

Rather than providing a dedicated instruction to end the execution of word in Forth and to return the caller's program flow, the N1 allows to perform this operation in parallel to the execution of any of its instructions. Each [opcode](#) contains a bit (bit 15) to indicate, that the current instruction is the last operation of the current word. If this bit is set, the program flow will resume at the calling word as soon as the operation is performed.

As shown in [Figure 2-1](#), bit 15 is also distinction between the encoding of [jump](#) and of [call](#) instructions. Considering that the last [call](#) in a word definition can be optimized to a [jump](#), bit 15 can be regarded as the termination bit for [call](#) instructions as well.

For a Forth compiler, this means that the semi-colon (;) always translates to setting bit 15 of the last instruction.

## 2.2 Jump Instructions

[Jump](#) instructions transfer the program flow to any address location within the supported 128KB program space. [Jump](#) instructions consume an absolute destination address which can either be placed on the top of the [parameter stack](#) or encoded into the opcode of the instruction (only for destination addresses < 0x3FFF).

## 2.3 Call Instructions

[Call](#) instructions temporarily transfer the program flow to any address location within the supported 128KB program space, while pushing a return address onto the [return stack](#). [Call](#) instructions consume an absolute destination address which can either be placed on the top of the [Parameter stack](#) or encoded into the opcode of the instruction (only for destination addresses < 0x3FFF).

## 2.4 Conditional Branches

[Conditional branches](#) invoke a change of program flow depending on the argument at the [top](#) of the [parameter stack](#). If it is zero, then the branch is taken. The branch destination is a [relative address](#), encoded into the opcode of the instruction in the range of  $\pm 8\text{KB}$ .

## 2.5 Literals

Signed integer [literals](#) of 12-bit length can be pushed onto the [parameter stack](#) within a single instruction. For larger integers a supplemental [ALU](#) instruction is required. (see encoding 11100 in [Table 2-1](#))

## 2.6 ALU Instructions

[ALU](#) instructions perform an operation on two [cell](#) values, resulting in a new double [cell](#) value. The result can either be placed entirely onto the [parameter stack](#), or truncated, discarding the most significant [cell](#). The first operand is always taken from the [parameter stack](#). The second operand can either be taken from the [parameter stack](#) or encoded into the opcode of the instruction. In the latter case, the interpretation of the embedded 5-bit value depends on the operation. The [immediate](#) value

is interpreted as either an unsigned ( $uimm$ ), a sign extended ( $simm$ ), or an offsetted ( $oimm$ ) integer value:

$$\begin{aligned}
 uimm &= \text{opcode}[4:0] \\
 simm &= \begin{cases} \text{opcode}[4:0], & \text{if } \text{opcode}[4:0] < 16 \\ \text{opcode}[4:0] - 32, & \text{if } \text{opcode}[4:0] \geq 16 \end{cases} \\
 oimm &= \text{opcode}[4:0] - 16
 \end{aligned}$$

Table 2-1 lists the supported ALU operations.

Table 2-1: ALU operations

Encoding	Operation	( x1 - d )	( x1 x2 - d )
00000	Sum	$x1 + uimm$	$x2 + x1$
00001	Absolute value	$oimm + \text{ABS}(x1)$	$x1 + \text{ABS}(x2)$
00010	Difference	$x1 - uimm$	$x2 - x1$
00011	Difference	$oimm - x1$	$x1 - x2$
00100	Unsigned minimum value	$\text{UMIN}(x1, uimm)$	$\text{UMIN}(x2, x1)$
00101	Signed maximum value	$\text{MAX}(oimm, x1)$	$\text{MAX}(x1, x2)$
00110	Unsigned maximum value	$\text{UMAX}(x1, uimm)$	$\text{UMAX}(x2, x1)$
00111	Signed minimum value	$\text{MIN}(oimm, x1)$	$\text{MIN}(x1 > x2)$
01000	Equals comparison	$x1 = uimm?$	$x2 = x1?$
01001	Equals comparison	$oimm = x1?$	$x1 = x2?$
01010	Not-equals comparison	$x1 \neq uimm?$	$x2 \neq x1?$
01011	Not-equals comparison	$oimm \neq x1?$	$x1 \neq x2?$
01100	Unsigned lower-than comparison	$x1 < uimm?$	$x2 < x1?$
01101	Signed lower-than comparison	$oimm < x1?$	$x1 < x2?$
01110	Unsigned greater-than comparison	$x1 > uimm?$	$x2 > x1?$
01111	Signed greater-than	$oimm > x1?$	$x1 > x2?$
10000	Unsigned product	$x1 * uimm$	$x2 * x1$
10001	Unsigned product	$x1 * simm$	$x2 * x1$
10010	Signed product	$x1 * uimm$	$x2 * x1$
10011	Signed product	$x1 * simm$	$x2 * x1$
10100	Logic AND	$x1 \wedge simm$	$x2 \wedge x1$
10101	Logic XOR	$x1 \oplus simm$	$x2 \oplus x1$
10110	Logic OR	$x1 \vee uimm$	$x2 \vee x1$
10111	Reserved		
11000	Logic right shift	$x1 \gg uimm$	$x2 \gg x1$
11001	Logic left shift	$x1 \ll uimm$	$x2 \ll x1$
11010	Arithmetic right shift	$x1 \gg uimm$	$x2 \gg x1$
11011	Reserved		
11100	Set upper bits of a literal value	$simm, x1[11:0]$	$simm, x2[11:0]$
11101	Reserved		
11110	Reserved		
11111	Reserved		

## 2.7 Stack Instructions

The N1's stack instruction aims at efficiently implementing common stack operations of the [Forth](#) language, while only implementing the essential data paths, which are needed for plain push and pull operations.

The opcode of the stack instruction contains a 10-bit field to specify a transition pattern of the upper [cells](#) of the [parameter stack](#) and the [return stack](#). The structure transition pattern is shown in [Figure 2-2](#).

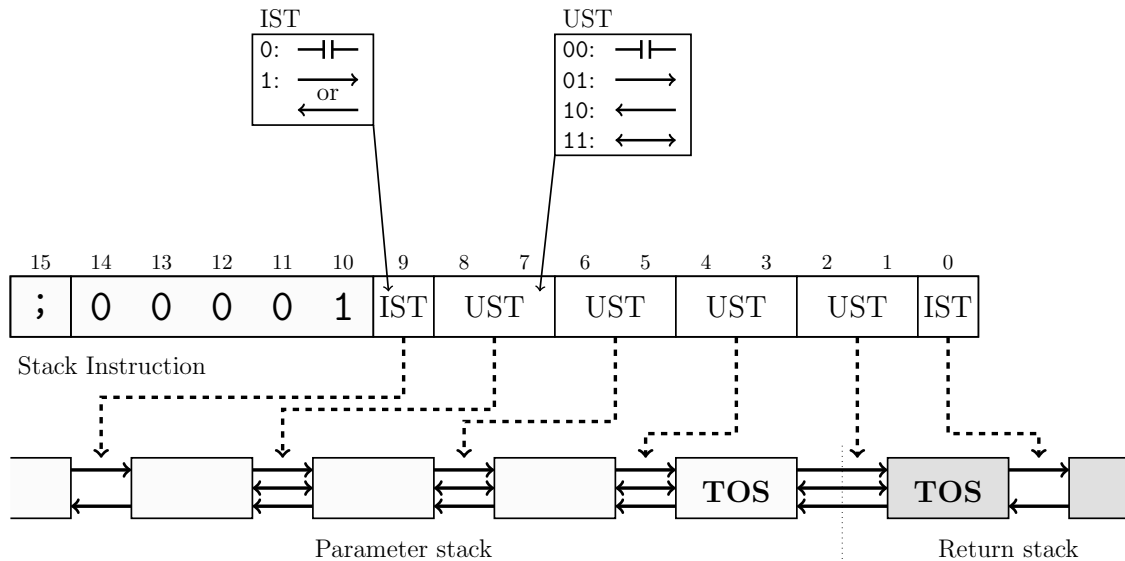


Figure 2-2: Transition encoding of stack instructions

The stack instruction contains four **UST** fields which control the data movement within the upper four [cells](#) of the [parameter stack](#) and the top [cell](#) of the [return stack](#). Each **UST** field determines the direction of data transfer between two neighboring stack [cells](#). Four options are selectable:

- No data transfer
- Data transfer upwards (or towards the [return stack](#))
- Data transfer downwards (or towards the [parameter stack](#))
- Data exchange between two stack [cells](#)

It is possible to put the **UST** fields into a combination which would trigger a data transfer of two source [cells](#) to a single destination [cell](#). In these cases, the resulting data in the destination [cell](#) is a logic OR of all sources.

The two remaining **IST** fields in the stack instruction control the data movement of the [lower stacks](#). Two options are selectable:

- No data transfer
- Data shift throughout the entire [intermediate stack](#). The direction is determined by the data movement of the lowest cell of the [upper stack](#).

[Table 2-2](#) shows how [stack](#) operations in [Forth](#) are mapped N1 instructions.

Table 2-2: Common stack operations

Word	Description	Transitions	Opcode
DROP	( x - )		0x06A8
DUP	( x - x x )		0x0750
SWAP	( x1 x2 - x2 x1 )		0x0418
OVER	( x1 x2 - x1 x2 x1 )		0x0758
NIP	( x1 x2 - x2 )		0x06A0
TUCK	( x1 x2 - x2 x1 x2 )		0x0750 0x0460
ROT	( x1 x2 x3 - x2 x3 x1 )		0x0460 0x0418
-ROT	( x1 x2 x3 - x3 x1 x2 )		0x0418 0x0460
RDROP	( R: x - )		0x0401
RDUP	( R: x - x x )		0x0407 0x0406
>R	( x - ) ( R: - x )		0x06AB
R@	( - x ) ( R: x - x )		0x0754
R>	( - x ) ( R: x - )		0x0755
2DROP	( x1 x2 - )		0x06A8 0x06A8
2DUP	( x1 x2 - x1 x2 x1 x2 )		0x0758 0x0758
2SWAP	( x1 x2 x3 x4 - x4 x3 x1 x2 )		0x0460 0x0598 0x0460
2OVER	( x1 x2 x3 x4 - x1 x2 x3 x4 x1 x2 )		0x0780 0x0460 0x0798 0x0460
2NIP	( x1 x2 x3 x4 - x3 x4 )		0x06A0 0x06A0





...continued

Table 2-2: Common stack operations

Word	Description	Transitions	Opcode
2TUCK	( x1 x2 x3 x4 – x3 x4 x1 x2 x3 x4 )		0x046B 0x0587 0x0418 0x0460 0x0755 0x0755
2ROT	( x1 x2 x3 x4 x5 x6 – x3 x4 x5 x6 x1 x2 )		0x06AB 0x0580 0x06AB 0x0598 0x0755 0x0598 0x0755 0x0598 0x0460
-2ROT	( x1 x2 x3 x4 x5 x6 – x5 x6 x1 x2 x3 x4 )		0x0460 0x0598 0x06AB 0x0598 0x06AB 0x0598 0x0755 0x0418 0x0755
2RDROP	( R: x1 x2 – )		0x0401 0x0401
2RDUP	( R: x1 x2 – x1 x2 x1 x2 )		0x0755 0x0757 0x06AB 0x06AB
2>R	( x1 x2 – ) ( R: – x1 x2 )		0x06AB 0x06AB

...continued

Table 2-2: Common stack operations

Word	Description	Transitions	Opcode
2R@	$(-x1\ x2)$ $(R: x1\ x2 - x1\ x2)$		0x0755
			0x0757
2R>	$(-x1\ x2)$ $(R: x1\ x2 -)$		0x0755
			0x0755

## 2.8 Memory Access Instructions

Memory access instructions perform read or write accesses to the system's 128KB address space. Data is solely accessed in 16-bit entities. Accesses to a 511B window in the main address space, can be done through an immediate addressing. This offers faster access to frequently used system variables.

## 2.9 Control Instructions

The N1 implements two types of control instructions to manipulate the internal state of the CPU. The first type are concurrent control instructions. These don't consume input from the stacks, nor do they produce a return value. These instructions perform multiple tasks within one bus cycle. [Table 2-3](#) shows the set of concurrent control instructions and their encoding.

Table 2-3: Concurrent Control Instructions

Encoding	Action
0b0000001xxxxxx1	Enable interrupts
0b0000001xxxxxx10	Disable interrupts
0b0000001xxxxx1xx	Enable exceptions
0b0000001xxxx10xx	Disable exceptions
0b0000001xxx1xxxx	Reset parameter stack
0b0000001xx1xxxxx	Reset return stack

The second type of control instructions trigger an internal sequence of actions and consume multiple clock cycles of execution time. [Table 2-4](#) lists the encoding of these complex control instructions.

Table 2-4: Non-concurrent control instruction encoding

Encoding	Instruction
0x00FF	Fetch parameter stack pointer (= number of cells) (- +n)

...continued

Table 2-4: Non-concurrent control instruction encoding

Encoding	Instruction
0x00FE	Store parameter stack pointer ( +n - )
0xb00FD	Fetch return stack pointer (= number of cells) ( - +n )
0x00FC	Store return stack pointer ( +n - )

### 3 ANS Forth Words

The N1 processor aims at executing Forth code in an efficient way. [Table 3-1](#) provides a list of standard ANS Forth[1] words which can be directly mapped to N1 instructions.

Table 3-1: ANS Forth words

Word	Stack	Description	Opcode
!	( x addr - )	Store x at addr	0x02FF
*	( n1 u1 n2 u2 - n3 u3 )	Multiply n1 u1 by n2 u2	0x0E00
+	( n1 u1 n2 u2 - n3 u3 )	Add n1 u1 to n2 u2	0x0C00
+	( n1 u1 a-adr - )	Add n1 u1 to the cell at addr	0x0403 0x03FF 0x0C00 0x0755 0x02FF
-	( n1 u1 n2 u2 - n3 u3 )	Subtract n2 u2 from n1 u1	0x0C40
0<	( n - flag )	Test if n is negative	0x0DF0
0<>	( x - flag )	Test if x is not zero	0x0D70
0>	( n - flag )	Test if n is greater than zero	0x0DB0
0=	( x - flag )	Test if x is not zero	0x0D30
1+	( n1 u1 - n2 u2 )	Increment n1 u1	0x0C01
1-	( n1 u1 - n2 u2 )	Decrement n1 u1	0x0C1F
2!	( x1 x2 addr - )	Store x2 at addr and x1 at addr+1	0x0750 0x0460 0x02FF 0x0C01 0x02FF
2*	( x1 - x2 )	Shift x1 one bit towards the <a href="#">MSB</a>	0x0F41
2/	( x1 - x2 )	Shift x1 one bit towards the <a href="#">LSB</a> , while the <a href="#">MSB</a> remains unchanged	0x0F41
2@	( addr - x1 x2 )	Fetch x2 from addr and x1 at addr+1	0x0750 0x0C01 0x03FF 0x0418 0x03FF
2DROP	( x1 x2 - )	Drop cell pair x1 x2	0x06A8 0x06A8
2DUP	( x1 x2 - x1 x2 x1 x2 )	Duplicate cell pair x1 x2	0x0758 0x0758
2OVER	( x1 x2 x3 x4 - x1 x2 x1 x2 x3 x4 x1 x2 )	Copy cell pair x1 x2 to the <a href="#">TOS</a>	0x0750 0x0460 0x0789 0x0460
2>R	( x1 x2 - ) (R: - x1 x2 )	Shift cell pair x1 x2 to the <a href="#">return stack</a>	0x06AB 0x06AB
2R>	( - x1 x2 ) (R: x1 x2 - )	Shift cell pair x1 x2 to the <a href="#">parameter stack</a>	0x0755 0x0755

...continued



Table 3-1: ANS Forth words

Word	Stack	Description	Opcode
2R0	( - x1 x2 ) (R: x1 x2 - x1 x2 )	Copy cell pair x1 x2 to the <a href="#">parameter stack</a>	0x0755 0x0757
2ROT	( x1 x2 x3 x4 x4 x5 x6 - x3 x4 x5 x6 x1 x2 )	Rotate three cell pairs	0x06AB 0x0580 0x06AB 0x0598 0x0755 0x0598 0x0755 0x0598 0x0460
2SWAP	( x1 x2 x3 x4 - x3 x4 x1 x2 )	Swap two cell pairs	0x0460 0x0598 0x0460
;	( - ) (R: addr - )	Return to the calling word	0x8400
<	( n1 n2 - flag )	Test if n1 is lower than n2	0x0DA0
<>	( x1 x2 - flag )	Test if x1 is different than x2	0x0D40
=	( x1 x2 - flag )	Test if x1 equals x2	0x0D00
>	( n1 n2 - flag )	Test if n1 is greater than n2	0x0DE0
>R	( x - ) (R: - x )	Shift x on to the <a href="#">return stack</a>	0x06AB
?DUP	( x - 0 x x )	Duplicate x if it is not zero	0x0750 0x0D30 0x2001 0x06A8
@	( addr - x )	Fetch x from addr	0x03FF
ABS	( n - u )	Absolute vale of n	0x0C30
AND	( x1 x2 - x3 )	Bitwise logic AND of x1 and x2	0x0E80
BL	( - char )	Space character	0x1020
CELL+	( addr1 - -addr2 )	Increment addr1	0x0C01
DEPTH	( - +n )	+n is the number of cells on the <a href="#">parameter stack</a> without +n	0x00FF
DROP	( x - )	Drop x from the /glsps	0x06A8
DUP	( x - x x )	Duplicate x	0x0750
EXECUTE	( i*x xt - j*x )	Execute xt	0x7FFF
FALSE	( - false )	FALSE flag	0x1000
I	( - n u ) ( R: n u - n u )	Copy the innermost loop index n u onto the <a href="#">parameter stack</a>	0x0754
INVERT	( x1 - x2 )	Bitwise inverse of x1	0x0EBF
J	( - n u ) ( R: x n u - x n u )	Copy the next-outer loop index n u onto the <a href="#">parameter stack</a>	0x0755 0x0407
LSHIFT	( x1 u - x2 )	Shift x1 u bits towards the <a href="#">MSB</a>	0x0F20
M*	( n1 n2 - d )	Multiply n1 by n2	0x0A40
M+	( n1 n2 - d )	Add n1 to n2	0x0800
MAX	( n1 n2 - n3 )	n3 is the greater of n1 and n2	0x0CA0
MIN	( n1 n2 - n3 )	n3 is the lesser of n1 and n2	0x0CE0
NEGATE	( n1 - n2 )	n2 is the two's complement of n1	0x0C70

...continued

Table 3-1: ANS Forth words

Word	Stack	Description	Opcode
NIP	( x1 x2 – x2 )	Drop x1	0x06A0
OR	( x1 x2 – x3 )	Bitwise logic OR of x1 and x2	0x0EC0
OVER	( x1 x2 – x1 x2 x1 )	Copy x1 to the <a href="#">TOS</a>	0x0758
R>	( – x ) (R: x – )	Shift x to the <a href="#">parameter stack</a>	0x0755
R@	( – x ) (R: x – x )	Copy x to the <a href="#">parameter stack</a>	0x0754
RSHIFT	( x1 u – x2 )	Shift x1 u bits towards the <a href="#">LSB</a>	0x0F00
ROT	( x1 x2 x3 – x2 x3 x1 )	Rotate the three topmost cells	0x0460 0x0418
S>D	( n – d )	Sign-extend n	0x0A41
SWAP	( x1 x2 – x2 x1 )	Swap x1 and x2	0x0418
TRUE	( – true )	TRUE flag	0x1FFF
TUCK	( x1 x2 – x2 x1 x2 )	Copy x1 below x2	0x0750 0x0460
U<	( u1 u2 – flag )	Test if u1 is lower than u2	0x0DC0
U>	( u1 u2 – flag )	Test if u1 is greater than u2	0x0D80
UM*	( u1 u2 – d )	Multiply u1 by u2	0x0A00
XOR	( x1 x2 – x3 )	Bitwise logic XOR of x1 and x2	0x0EA0

## 4 Stacks

The N1 operates with two stacks: the [parameter stack](#) to perform data transactions and the [return stack](#) to manage the program flow. As illustrated in [Figure 4-1](#), each of these stacks consists of three segments: the [upper stack](#), the [intermediate stack](#), and the [lower stack](#).

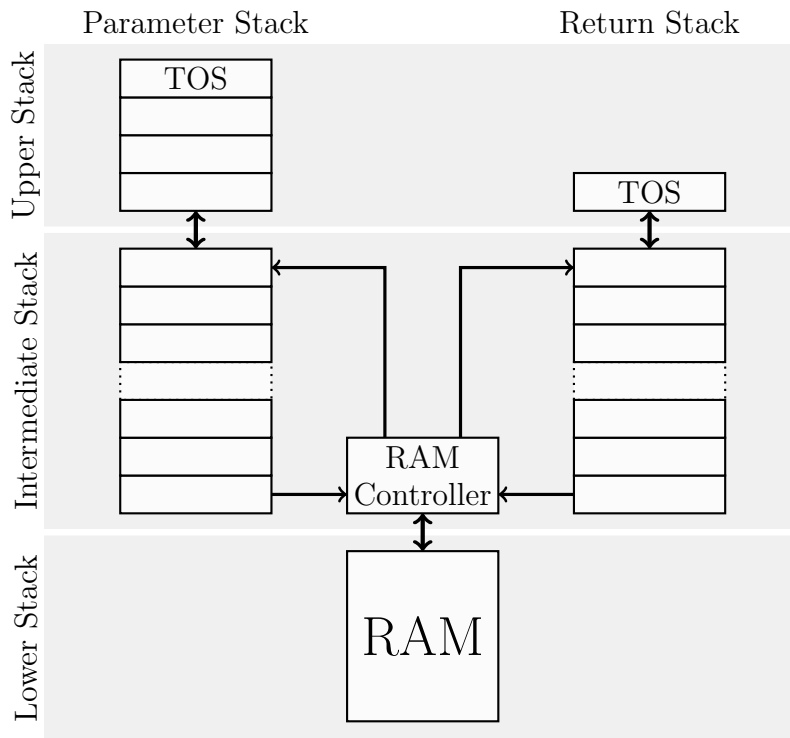


Figure 4-1: Stack Architecture

### 4.1 Parameter Stack

The [upper parameter stack](#) holds the four topmost data entries. Its purpose is to perform stack and [ALU](#) operations (see [Section 2.7 “Stack Instructions”](#) and [Section 2.6 “ALU Instructions”](#)). When the capacity of the [upper stack](#) is exceeded, older data entries are shifted to the [intermediate stack](#).

The [intermediate stack](#) serves as a buffer between the [upper stack](#) and the [lower stack](#), which resides in [RAM](#). The purpose of the [intermediate stack](#) is to minimize [RAM](#) traffic to and from the [lower stack](#). Push operations to the [intermediate stack](#) are only propagated to the [lower stack](#) when the buffer capacity is exceeded. Pull operations are only propagated when the [intermediate stack](#) is empty. [Stack](#) fluctuations within the [intermediate stack](#)’s capacity are not visible to the [lower stack](#).

The [lower stack](#) is a region of the [RAM](#), which is managed by a memory controller that is shared by the [parameter stack](#) and the [return stack](#). Within the [RAM](#), both stacks will grow towards each other. Moving cell content from one stack to the other ([>R](#) or [R>](#)) will never lead to a stack overflow.

## 4.2 Return Stack Stack

The [upper stack](#) of the [parameter stack](#) has the capacity of one [cell](#). The [intermediate stack](#) and [lower stack](#) are identical to the ones of the [parameter stack](#).

## 5 Reset, Exceptions, and Interrupts

There are three hardware mechanisms in the N1 processor, which can stop the ongoing program flow in order to react to an urgent hardware condition: Reset, Exceptions and Interrupts.

### 5.1 Reset

A reset puts the entire sequential logic of the N1 into a defined initial state. The [return stack](#) becomes completely cleared and the [parameter stack](#) is initialized to hold exactly one cell, containing the reset indicator 0x0000 (see [Table 5-1](#)). After every reset, program execution will begin at address 0x0000. Any context of the previous program flow is lost. Resets are generated by the system's hardware and occur at least once during power-up.

### 5.2 Exceptions

Exceptions are triggered by error conditions and allow the software to restore the functionality of the system. There are five error conditions, which can be detected by the N1 hardware:

#### [Parameter stack overflow](#)

A [parameter stack](#) overflow occurs when the capacity of the lower stack's RAM is exceeded (excluding a little margin, which is required for the error handling).

#### [Return stack stack underflow](#)

A [parameter stack](#) underflow occurs when an instruction requires more arguments than available on the [stack](#) and when a stack instruction would result in non-continuous filling of the stack.

#### [Return stack overflow](#)

A [return stack](#) overflow occurs when the capacity of the lower stack's RAM is exceeded (excluding a little margin, which is required for the error handling).

#### [Return stack underflow](#)

A [return stack](#) underflow occurs when an instruction requires more arguments than available on the [return stack](#).

#### [Address out of range](#)

This error condition indicates a memory access to a restricted address. This can either be caused by an instruction fetch or a data access

In any of these cases, the N1 processor will push a [throw code](#) (see [Table 5-1](#)) onto the [parameter stack](#) and proceed with code execution at address 0x0000. The [return stack](#) and the lower content of the [parameter stack](#) remain untouched. The context of the previous program execution is not reserved. To avoid reoccurrence of error conditions during the execution of the handler routine, exceptions are temporarily disabled after detection. Exceptions must then be reenabled by a control instruction (see [Table 2-3](#)) when the error is resolved. The [throw codes](#) listed in [Table 5-1](#) comply with the exception word set of the ANS Forth standard [1].

Table 5-1: Throw codes

Throw Code	Condition
0x0000 (0)	Reset
0xFFFFD (-3)	Parameter stack overflow
0xFFFFC (-4)	Parameter stack underflow
0xFFFFB (-5)	Parameter stack overflow
0xFFFFA (-6)	Parameter stack underflow
0xFFFF7 (-9)	Invalid memory address

The five hardware exceptions can be easily complemented by user defined software exceptions. Software exceptions can be thrown by pushing a unique [throw code](#) onto the [parameter stack](#) and performing a [jump](#) to address 0x0000. Hardware and software exceptions can then be handled by a common exception handler routine.

### 5.3 Interrupts

Interrupts are service requests which are generated by the peripheral hardware. They cause a temporary interruption of the ongoing program flow. When an interrupt occurs, the program counter is saved to the [return stack](#) and an interrupt service routine is executed. The location of the interrupt service routine is determined by the system's interrupt controller hardware. Further interrupts are automatically disabled during the execution of the interrupt service routine and must be manually reenabled by a control instruction (see [Table 2-3](#)) before resuming the interrupted program flow.

## 6 Integration Guide

This section outlines the interfaces and configurations of the N1 processor for system integration.

### 6.1 Integratation Parameters

The N1 processor supports six Verilog integration parameters to configure the design for application specific needs:

#### SP\_WIDTH

Stack pointer width.

This parameter determines the address width of the [lower stack](#). Values in the range of 5 to 16 are valid. The default value is 12.

#### IPS\_DEPTH

Depth of the intermediate parameter stack.

This parameter determines the number of [cells](#) in the [intermediate stack](#) of the [parameter stack](#). Any value larger than 2 is valid. The default value is 8. The purpose of the [intermediate stack](#) is to conceal fluctuations in stack usage to the [lower stack](#). The optimal value should be derived from the application use case.

#### IRS\_DEPTH

Depth of the intermediate return stack.

This parameter determines the number of [cells](#) in the [intermediate stack](#) of the [return stack](#). Any value larger than 2 is valid. The default value is 8. The purpose of the [intermediate stack](#) is to conceal fluctuations in stack usage to the [lower stack](#). The optimal value should be derived from the application use case.

#### PBUS\_AADR\_OFFSET

Offset for direct [jump](#) or [call](#) addressing.

This parameter determines the location of the 32KB window for [jumps](#) and [calls](#) with [direct addressing](#). The default value is 0x0000.

#### PBUS\_MADR\_OFFSET

Offset for direct data accesses.

This parameter determines the location of the 511B window for memory I/O with [direct addressing](#). This window should cover commonly used [Forth](#) variables. The default value is 0xFFFF.

#### PS\_RS\_DIST

Safety distance between the [parameter stack](#) and the [return stack](#).

Recovering from an exception requires some free [stack](#) space. This parameter determines the remaining [stack](#) space when a [stack](#) overflow exception is thrown. The default value is 22. The optimal value depends on the requirements of the exception handler software.

### 6.2 Interfaces

The N1 processor provides four interfaces which must be connected at system level. A fifth one (see [Section 6.2.5 “Probe Signals”](#)) is only to be used for verification and debug purposes.

### 6.2.1 Clock and Resets

This interface provides clocks and resets for all sequential logic in the N1 design.

**clk\_i**

Single clock input.

This clock is used for all interfaces as well as all internal sequential logic.

**async\_rst\_i**

Asynchronous reset input.

This active high reset input may assert asynchronously, but must deassert synchronously. This signal is not required if a synchronous reset (**sync\_rst\_i**) is implemented. If unused, this input must be tied to 0.

**sync\_rst\_i**

Synchronous reset input.

This active high reset input must assert and deassert synchronously. This signal is not required if an asynchronous reset (**async\_rst\_i**) is implemented. If unused, this input must be tied to 0.

### 6.2.2 Program Bus

This interface connects the N1 to the main memory. All signals comply to the [Wishbone](#) protocol [2].

**pbus\_cyc\_o**

Cycle indicator output.

This output signal corresponds to signal **CYC\_0** of the Wishbone specification [2].

**pbus\_stb\_o**

Strobe output.

This output signal corresponds to signal **STB\_0** of the Wishbone specification [2].

**pbus\_we\_o**

Write enable output.

This output signal corresponds to signal **WE\_0** of the Wishbone specification [2].

**pbus\_adr\_o**

Address bus.

These output signals correspond to bus **ADR\_0** of the Wishbone specification [2].

**pbus\_dat\_o**

Write data bus.

These output signals correspond to bus **DAT\_0** of the Wishbone specification [2].

**pbus\_tga\_cof\_jump\_o**

Change of flow indicator.

This output signal corresponds to bus **TGA\_0** of the Wishbone specification [2]. It indicates, that the current bus access was caused by a [jump](#) instruction. This information may be used to trace the program flow.

**pbus\_tga\_cof\_cal\_o**

Change of flow indicator.

This output signal corresponds to bus **TGA\_0** of the Wishbone specification [2]. It indicates, that the current bus access was caused by either a [call](#) instruction or an interrupt service request. This information may be used to trace the program flow.



**pbus\_tga\_cof\_bra\_o**

Change of flow indicator.

This output signal corresponds to bus **TGA\_0** of the Wishbone specification [2].

It indicates, that the current bus access was caused by a [conditional branch](#) instruction. This information may be used to trace the program flow.

**pbus\_tga\_cof\_eow\_o**

Change of flow indicator.

This output signal corresponds to bus **TGA\_0** of the Wishbone specification [2].

It indicates, that the current bus access was caused by a return from a [call](#).

This information may be used to trace the program flow.

**pbus\_ack\_i**

Acknowledge input.

This input signal corresponds to signal **ACK\_I** of the Wishbone specification [2].

If unused, this input must be tied to 1.

**pbus\_err\_i**

Error indicator input.

This input signal corresponds to signal **ERR\_I** of the Wishbone specification [2].

It informs the N1 processor, that the current address exceeds the valid range of the connected memory system. If unused, this input must be tied to 0.

**pbus\_stall\_i**

Pipeline stall input.

This input signal corresponds to signal **STALL\_I** of the Wishbone specification [2]. If unused, this input must be tied to 0.

**pbus\_dat\_i**

Read data bus.

These input signals correspond to bus **DAT\_I** of the Wishbone specification [2].

**6.2.3 Stack Bus**

This interface connects the N1 to the stack memory. It is expected that the **SP\_WIDTH** parameter (see [Section 6.1 “Integration Parameters”](#)) matches the implemented memory size. Therefore no **ERR\_I** input is needed in this interface. All signals comply to the [Wishbone](#) protocol [2].

**sbus\_cyc\_o**

Cycle indicator output.

This output signal corresponds to signal **CYC\_0** of the Wishbone specification [2].

**sbus\_stb\_o**

Strobe output.

This output signal corresponds to signal **STB\_0** of the Wishbone specification [2].

**sbus\_we\_o**

Write enable output.

This output signal corresponds to signal **WE\_0** of the Wishbone specification [2].

**sbus\_adr\_o**

Address bus.

These output signals correspond to bus **ADR\_0** of the Wishbone specification [2].

**sbus\_dat\_o**

Write data bus.

These output signals correspond to bus DAT\_O of the Wishbone specification [2].

**sbus\_tga\_ps\_o**

[Parameter stack](#) access indicator.

These output signals correspond to bus TGA\_O of the Wishbone specification [2].

It indicates, that the current bus access is associated with the [parameter stack](#).

**sbus\_tga\_rs\_o**

[Return stack](#) access indicator.

These output signals correspond to bus TGA\_O of the Wishbone specification [2].

It indicates, that the current bus access is associated with the [return stack](#).

**sbus\_ack\_i**

Acknowledge input.

This input signal corresponds to signal ACK\_I of the Wishbone specification [2].

If unused, this input must be tied to 1.

**sbus\_stall\_i**

Pipeline stall input.

This input signal corresponds to signal STALL\_I of the Wishbone specification [2]. If unused, this input must be tied to 0.

**sbus\_dat\_i**

Read data bus.

These input signals correspond to bus DAT\_I of the Wishbone specification [2].

**6.2.4 Interrupt Interface**

This interface connects an optional interrupt controller to the N1 processor.

**irq\_ack\_o**

Interrupt acknowledge.

This output signal asserts for one clock cycle, whenever the current interrupt is serviced. It may be used for automatic flag clearing.

**irq\_req\_i**

Interrupt request.

Any non-zero value driven to this bus interface is interpreted as interrupt request. The value determines the start address of the interrupt service routine that is to be executed by the N1 processor. This bus must be tied to 0x0000 if no interrupt controller is connected.

**6.2.5 Probe Signals**

This interface propagates all internal states of the N1 processor to the outside. It is solely intended for verification and debug purposes and should be left unconnected for system integration. The signals in this interface are specific to the internal implementation of the N1 processor and may change with every revision.

### 6.3 Target Specific Design Files

All adder and multiplier logic of the N1 design is located in a single [Verilog](#) module called `N1_dsp`. A synthesizable implementation of this module, can be found in the file `rtl/verilog/N1_dsp_synth.v`. If desired, this file can be replaced by one containing a alternative implementation of the `N1_dsp` module. An example is given in in the file `rtl/verilog/N1_dsp_iCE40UP5K.v`. It contains a custom implementation for Lattice iCE40 FPGAs, utilizing four hard instantiated `SB_MAC16` macro cells.

## 7 Verification Status

The implementation of the N1 design is currently still ongoing. Verification has not yet begun.

## 8 Tool Summary

One of the main goals of the N1 project is to use a design and verification flow, based on open source EDA tools. [Table 8-1](#) summarizes the tools, used for this project.

Table 8-1: Tool Summary

Tool	Version	Usage
Verrilator <a href="#">[4]</a>	3.874	Linting
Icarus Verilog <a href="#">[6]</a>	0.9.7	Linting
Yosys <a href="#">[8]</a>	0.7+627	Linting, Formal Verification
SymbiYosys <a href="#">[7]</a>	Sep. 12, 2018	Formal Verification
GTKWave <a href="#">[3]</a>	3.3.95	Waveform Viewer
Verilog-Perl <a href="#">[5]</a>	3.418-1	Gereration of design data for GTKWave <a href="#">[3]</a>

## 9 Glossary

;

End of a [word](#) definition in [Forth](#).

### ALU

Arithmetic Logic Unit.

### call

A change of the program flow, where a return address is kept on the [return stack](#) (see [Section 2.3 “Call Instructions”](#)).

### cell

A data entity within a [stack](#).

### conditional branch

A change of the program flow without return option, only if a certain (non-zero) argument value is given (see [Section 2.4 “Conditional Branches”](#)).

### direct addressing

Addressmode, where the address is encoded into the [opcode](#) of an instruction

### Forth

Forth is an extensible stack-based programming language.

### immediate data

A data value, which is encoded into the [opcode](#) of an instruction

### indirect addressing

Address mode, where the address ist stored on the [parameter stack](#).

### intermediate stack

The section of the stack that serves as a buffer between the [lower stack](#) and the [upper stack](#). See [Section 4 “Stacks”](#).

### IST

A bit field in the stack instruction which contols data movement on the intermediate [parameter stack](#) or [return stack](#). The mnemonic stands for “**I**ntermediate **S**tack **T**ransition”.

### jump

A change of the program flow without return option (see [Section 2.2 “Jump Instructions”](#)).

### LIFO

A memory which is accessible in last in - first out order.

### literal

A fixed numerical value within the program code (see [Section 2.5 “Literals”](#)).

**lower stack**

The section of the stack which is stored in RAM. See [Section 4 “Stacks”](#).

**LSB**

The least significant bit.

**MSB**

The most significant bit.

**opcode**

Encoding of a machine instruction. Short for “operation code”.

**parameter stack**

A [LIFO](#) storage mainly for keeping call parameters and return values.

**RAM**

Random access memory.

**relative addressing**

Addressmode, where the address is given relative to the current position in the execution flow

**return stack**

A [LIFO](#) storage mainly for maintaining return addresses of [calls](#).

**stack**

A [LIFO](#) storage.

**throw code**

A unique identifier for each type of exception.

**TOS**

The top [cell](#) of a [stack](#).

**upper stack**

The section of the stack that contains the [TOS](#). It supports reordering of its storage [cell](#). See [Section 4 “Stacks”](#).

**UST**

A bit field in the stack instruction which controls data movement between two neighboring [cells](#) in the upper [parameter stack](#) or [return stack](#). The mnemonic stands for “Upper Stack Transition”.

**Verilog**

The hardware description language used for the N1 implementation.

**Von-Neumann-Architecture**

A computer architecture where instruction fetches and data I/O occur over the same memory interface.

**Wishbone**

An open bus prototocoll. see [\[2\]](#)

**word**

The term word refers to a callable code sequence in [Forth](#) terminology.



## 10 References

- [1] American National Standard for Information Systems, 1994.
- [2] Wishbone b4. [http://cdn.opencores.org/downloads/wbspec\\_b4.pdf](http://cdn.opencores.org/downloads/wbspec_b4.pdf), 2010.
- [3] BSI. Gtkwave. <http://gtkwave.sourceforge.net>.
- [4] Wilson Snyder. Verilator. <http://www.veripool.org/verilator>.
- [5] Wilson Snyder. Verilog-perl. <http://www.veripool.org/verilog-perl>.
- [6] Stephen Williams. Icarus verilog. <http://iverilog.icarus.com>.
- [7] Clifford Wolf. Symbiyosys. <https://github.com/cliffordwolf/SymbiYosys>.
- [8] Clifford Wolf. Yosys open synthesis suite. <http://www.clifford.at/yosys>.