# N1 Manual

Dirk Heisswolf

November 30, 2018

# Revision History

| Date | Change |
|---|---|
| November 30, 2018 | Pre-release |

# Contents

# List of Figures

# List of Tables

# 1 Glossary

**;**

End of a word definition in Forth.

**ALU**

Arithmetic Logic Unit.

**branch**

A change of the program flow without return option, where the destination is given as offset from the start point.

**byte**

An 8-bit data entity.

**call**

A change of the program flow, where a return address is kept on the return stack.

**cell**

A data entity within a stack.

**Forth**

Forth is a extensible stack-based programming language.

**intermediate stack**

The section of the stack, which 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**intermediate **S**tack **T**ransition".

**jump**

A change of the program flow without return option.

**LIFO**

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

**literal**

A fixed numerical value within the program code.

**lower stack**

The section of the stack which stored in RAM. See Section 4 "Stacks".

**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.

**return stack**

A LIFO storage mainly for maintaining return addresses of calls.

**stack**

A LIFO storage.

**TOS**

The top cell of a stack.

**upper stack**

The section of the stack, which contains the TOS. It supports reordering of its storage cell. See Section 4 "Stacks".

**UST**

A bit field in the stack instruction which contols data movement between two neighboring cells in the upper parameter stack or return stack. The mnemonic stands for "**U**pper **S**tack **T**ransition".

**word**

The term word is used in two different contexts throughout this document. It refers to either a 16-bit data entity or a callable code sequence in Forth terminology.

# 2 Overview

The N1 is a snall stack machine, inspired by the J1 Forth CPU[1]. Just like its paragon, the N1 is a 16-bit processor wich implements basic Forth words directly in hardware. However the N1 parts from the J1's simplistic design approach in in two ways:

- The N1 support a larger code space of up to 32KB. Therefore it has its own instruction set (see Section 3 "Instruction Set".

- The N1 implements its parameter and return stacks as shallow register stacks, which overflow into RAM. The overall depth of each stack is determined by the available RAM. (see Section 4 "Stacks".

# 3   Instruction Set

The intent of the N1's instruction set is to map most of the essential Forth words to single cycle instructions. Figure 3-1 illustrates the basic structure of the instructuion encoding.

End of
Word

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 14-bit absolute word address | | | | | | | | | | | | | | Jump |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 14-bit absolute word address | | | | | | | | | | | | | | Call |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| ; | 0 | 1 | 13-bit relative word address | | | | | | | | | | | | | Conditional branch |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| ; | 0 | 0 | 1 | 12-bit signed integer | | | | | | | | | | | | Literal |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| ; | 0 | 0 | 0 | 1 | $R/\overline{W}$ | 10-bit register address | | | | | | | | | | Register access |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| ; | 0 | 0 | 0 | 0 | 1 | Stack transition pattern | | | | | | | | | | Stack operation |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| ; | 0 | 0 | 0 | 0 | 0 | 1 | Instruction | | | | | | | | | I/O and control instructions |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| ; | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | Operator | | | Operand | | | | ALU operation (n – n) |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| ; | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | Operator | | | | | | | ALU operation (n n - n) |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| ; | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Operator | | | Operand | | | | ALU operation (n – n n) |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| ; | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Operator | | | | | | | ALU operation (n n - n n) |

Figure 3-1: Instruction encoding

## 3.1 Jump Instructions

Jump instructions transfer the program flow to any word location within the supported 32KB program space. Jump instructions use absoluteaddresses.

## 3.2 Call Instructions

Call instructions temporarily transfer the program flow to any word location within the supported 32KB program space, while pushing a return address onto the return stack call instructions use absolute addresses.

## 3.3 Return from a Call (;)

Rather than providing a dedicated instruction to end the execution of word in Forth and to return the program flow to its caller, 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 in the last operation in the current word. If this bit is set, the program flow will resume at the calling word as soon as the operationis performed.

As shown in Figure 3-1, bit 15 is also used to distinguish jump and call. Considering that the last call in a word definition can be optimized to a jump to the first instruction of the called word, bit 15 can ber regarded as termination bit for these instructions as well.

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

## 3.4 Conditional Branches

Conditional Branches invoke a change of program flow if the top of the parameter stack is not zero. Branches are relative to the location of the following instruction and range from 4095 word locations forward to 4096 word locations backward. A relative branch address of zero points to the subsequent instruction of the current one.

Conditional branches use relative addressing to simplify code reallocation in support of inlining.

## 3.5 Literals

Signed integer literals of 12-bit length can be pushed onto the parameter stack within a single instruction. For larger integers a supplemental TBD call is required.

## 3.6 Register Accesses

## 3.7 Stack Operation

The N1's stack instruction aims at efficiently implementing the essential stack operations in Forth only using the data pathes which needed for the stack's push and pull operations.

The opcode of the stack instruction contains a 10-bit wide field to specify a transition pattern of the upper cells of the parameter stack and the return stack. The structure transition patter is shown in Figure 3-2.

The stack instruction contains four UST fields which control the data transfer within the upper four cells of the parameter stack and the top of the return stack.
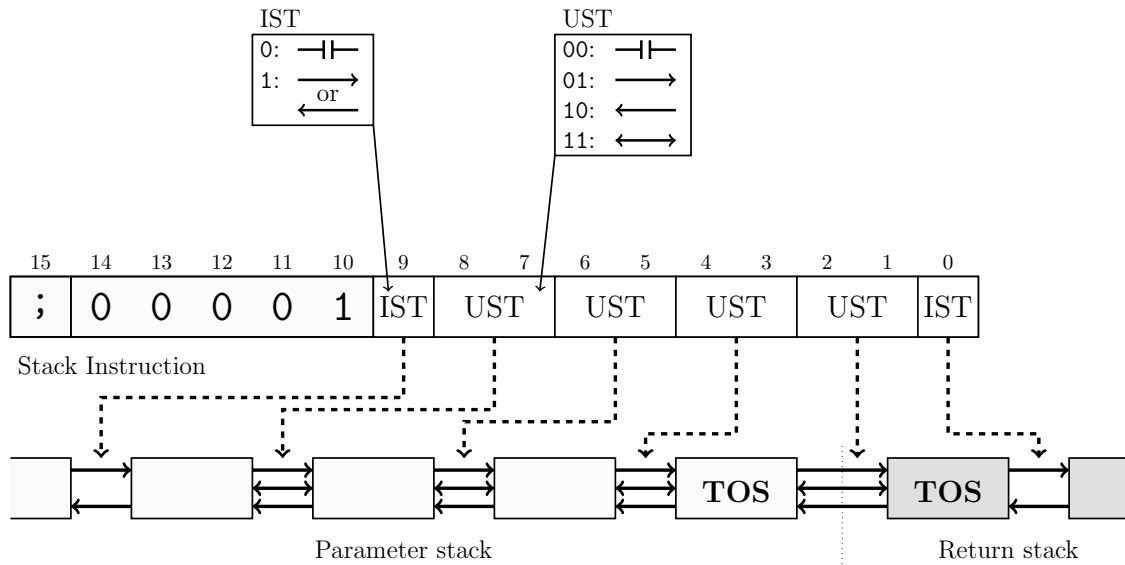
Figure 3-2: Transition encoding of stack instructions

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 desination cell. In these cases, the resulting data in the desination cell is undefined.

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 3-1 shows how stack operations in Forth are mapped N1 instructions.

Table 3-1: Common stack operations

| Word | Description | Transitions | | | Opcode |
|------|-------------|-------------|---|---|--------|
| DROP | ( x − ) |  | TOS | | 0x06A8 |
| DUP | ( x − x x ) |  | TOS | | 0x0750 |
| SWAP | ( x1 x2 − x2 x1 ) |  | TOS | | 0x0418 |

...continued

10

Table 3-1: Common stack operations

| Word | Description | Transitions | Opcode |
|------|-------------|-------------|--------|
| OVER | ( x1 x2 – x1 x2 x1 ) | *(diagram)* | 0x0758 |
| NIP | ( x1 x2 – x2 ) | *(diagram)* | 0x06A0 |
| TUCK | ( x1 x2 – x2 x1 x2 ) | *(diagram)* | 0x0750<br>0x0460 |
| ROT | ( x1 x2 x3 – x2 x3 x1 ) | *(diagram)* | 0x0460<br>0x0418 |
| -ROT | ( x1 x2 x3 – x3 x1 x2 ) | *(diagram)* | 0x0418<br>0x0460 |
| RDROP | ( R: x – ) | *(diagram)* | 0x0001 |
| RDUP | ( R: x – x x ) | *(diagram)* | 0x0007<br>0x0006 |
| >R | ( x – )<br>( R: – x ) | *(diagram)* | 0x06AB |
| R@ | ( – x )<br>( R: x – x ) | *(diagram)* | 0x0754 |
| R> | ( – x )<br>( R: x – ) | *(diagram)* | 0x0755 |
| 2DROP | ( x1 x2 – ) | *(diagram)* | 0x06A8<br>0x06A8 |
| 2DUP | ( x1 x2 – x1 x2 x1 x2 ) | *(diagram)* | 0x0758<br>0x0758 |
| 2SWAP | ( x1 x2 x3 x4 – x4 x3 x1 x2 ) | *(diagram)* | 0x0460<br>0x0598<br>0x0460 |
| 2OVER | ( x1 x2 x3 x4 – x1 x2 x3 x4 x1 x2 ) | *(diagram)* | 0x0780<br>0x0460<br>0x0798<br>0x0460 |
| 2NIP | ( x1 x2 x3 x4 – x3 x4 ) | *(diagram)* | 0x06A0<br>0x06A0 |
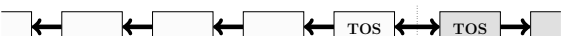
Table 3-1: Common stack operations

| Word | Description | Transitions | Opcode |
|------|-------------|-------------|--------|
| 2TUCK | ( x1 x2 x3 x4 – x3 x4 x1 x2 x3 x4 ) | | 0x046B<br>0x0487<br>0x0418<br>0x0460<br>0x0755<br>0x0755 |
| 2ROT | ( x1 x2 x3 x4 x5 x6 – x3 x4 x5 x6 x1 x2 ) | | 0x06AB<br>0x0580<br>0x06AB<br>0x0598<br>0x0755<br>0x0598<br>0x0755<br>0x0598<br>0x0460 |
| -2ROT | ( x1 x2 x3 x4 x5 x6 – x5 x6 x1 x2 x3 x4 ) | | 0x0460<br>0x0598<br>0x06AB<br>0x0598<br>0x06AB<br>0x0598<br>0x0755<br>0x0018<br>0x0755 |
| 2RDROP | ( R: x1 x2 – ) | | 0x0001<br>0x0001 |
| 2RDUP | ( R: x1 x2 – x1 x1 x1 x2 ) | | 0x0755<br>0x0757<br>0x06AB<br>0x06AB |
| 2>R | ( x1 x2 – )<br>( R: – x1 x2 ) | | 0x0000<br>0x0000 |

12

Table 3-1: Common stack operations

| Word | Description | Transitions | Opcode |
|:---:|:---:|:---:|:---:|
| 2R@ | ( – x1 x2 )<br>( R: x1 x2 – x1 x2 ) | | 0x0000<br>0x0000 |
| 2R> | ( – x1 x2 )<br>( R: x1 x2 – ) | | 0x0000<br>0x0000 |

## 3.8 I/O and Control Instructions

Byte

## 3.9 ALU operations

ALU

# 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 **??**, each of these stacks consists of three hardware components: the upper stack, the intermediate stack, and the lower stack.

# TBD

Figure 4-1: Instruction encoding

## 4.1   Parameter Stack

The upper stack of the parameter stack contains is four cells deep and contains the most recent data entries. It's purpose is to perform stack and ALU operations (see Section 3.7 "Stack Operation" and Section 3.9 "ALU operations"). When the capacity of the upper stack is exceeded, older data entries are transferred 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 operation to the intermediate stack are only propagated to the lower stack, when the buffer capacity is exceeded. Pull operations are onle propagated, when the intermediate stack is empty. Stack fluctuations within the buffer capacity are not visible to the lower stack.

The lower stack is a region of the RAM, which is managed by the memory controller of the intermediate stack.

## 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 similar to the ones of the parameter stack.

# References

[1] Menlo Park James Bowman, Willow Garage. J1: a small forth cpu core for fpgas. http://www.excamera.com/files/j1.pdf, 2010.