N1 Manual

Dirk Heisswolf

March 12, 2019

Revision History

Date	Change
March 12, 2019	Pre-release

Contents

List of Figures

List of Tables

1 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.

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.

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 5 "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 5 "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.

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 5 "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 refers to a callable code sequence in Forth terminology.

2 Overview

The N1 is a snall stack machine, inspired by the J1 Forth CPU[2]. 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 5 “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.

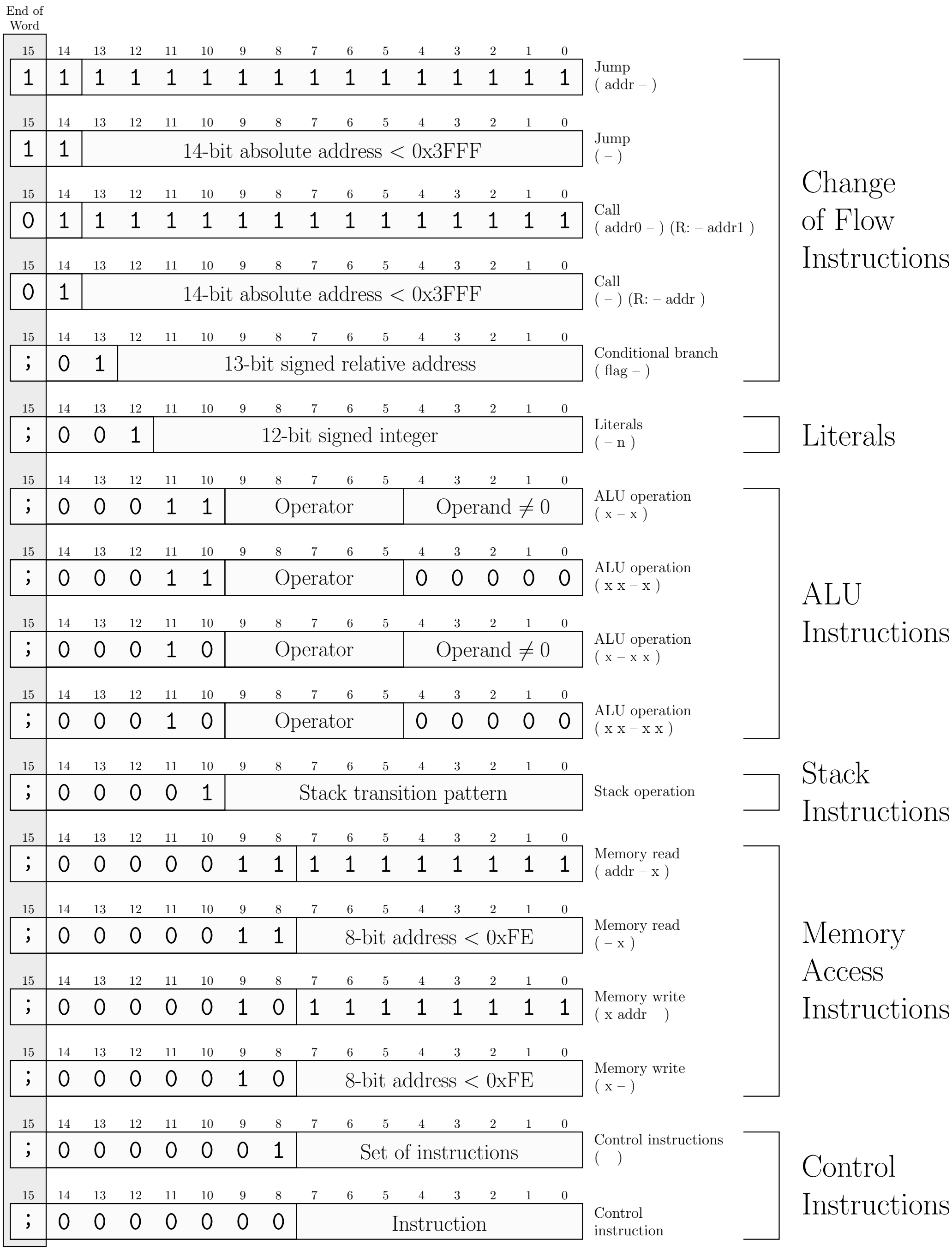


Figure 3-1: Instruction encoding

3.1 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.

3.2 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 be either placed on the top of the Parameter stack or encoded into the opcode of the instruction (only for destination addresses $<$ 0x3FFF).

3.3 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 be either placed on the top of the Parameter stack or encoded into the opcode of the instruction (only for destination addresses $<$ 0x3FFF).

3.4 Conditional Branches

Conditional branches invoke a change of program flow depending on an argument on the parameter stack. The branch destination cab be either an absolute address placed on the the top of the Parameter stack or relative relative address, encoded into the opcode of the instruction (only for destination addresses $<$ 0x1FFF).

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 ALU Instructions

ALU instructions perform an operation on two cell values, resulting in a new double cell value. The reult can be either 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 be either 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. It is either regarded as an unsigned (*uimm*), a sign extended (*simm*), or an offsetted (*oimm*) integer value:

$$uimm = \text{opcode}[4{:}0]$$

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

$$oimm = \text{opcode}[4{:}0] - 16$$

Table 3-1 lists the supported ALU operations.

Table 3-1: ALU operations

Encoding	**Operation**	**(x1 – d)**	**(x1 x2 – d)**
00000	Sum	x1 + $uimm$	x1 + x2
00001	Absolute value	$oimm$ + ABS(x1)	x2 + ABS(x1)
00010	Difference	x1 − $uimm$	x1 − x2
00011	Difference	$oimm$ − x1	x2 − x1
00100	Unsigned lower-than comparison	x1 < $uimm$?	x1 < x2?
00101	Signed greater-than comparison	$oimm$ < x1?	x2 < x1?
00110	Unsigned greater-than comparison	x1 > $uimm$?	x1 > x2?
00111	Signed lower-than comparison	$oimm$ > x1?	x2 > x1?
01000	Equals comparison	x1 = $uimm$?	x1 = x2?
01001	Equals comparison	$oimm$ = x1?	x2 = x1?
01010	Not-equals comparison	x1 ≠ $uimm$?	x1 ≠ x2?
01011	Not-equals comparison	$oimm$ ≠ x1?	x2 ≠ x1?
01100	Unsigned minimum	UMIN(x1, $uimm$)	UMIN(x1, x2)
01101	Signed maximum	MAX($oimm$, x1)	MAX(x2, x1)
01110	Unsigned maximum	UMAX(x1, $uimm$)	UMAX(x1, x2)
01111	Signed minimum	MIN($oimm$, x1)	MIN(x2, x1)
10000	Unsigned product	x1 ∗ $uimm$	x1 ∗ x2
10001	Unsigned product	x1 ∗ $simm$	x1 ∗ x2
10010	Signed product	x1 ∗ $uimm$	x1 ∗ x2
10011	Signed product	x1 ∗ $simm$	x1 ∗ x2
10100	Logic AND	x1 ∧ $simm$	x1 ∧ x2
10101	Logic XOR	x1 ⊕ $simm$	x1 ⊕ x2
10110	Logic OR	x1 ∨ $uimm$	x1 ∨ x2
10111	Reserved		
11000	Logic right shift	x1 ≫ $uimm$	x2 ≫ x1
11001	Logic left shift	x1 ≪ $uimm$	x2 ≪ x1
11010	Arithmetic right shift	x1 ≫ $uimm$	x2 ≫ 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		

3.7 Stack Instructions

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.

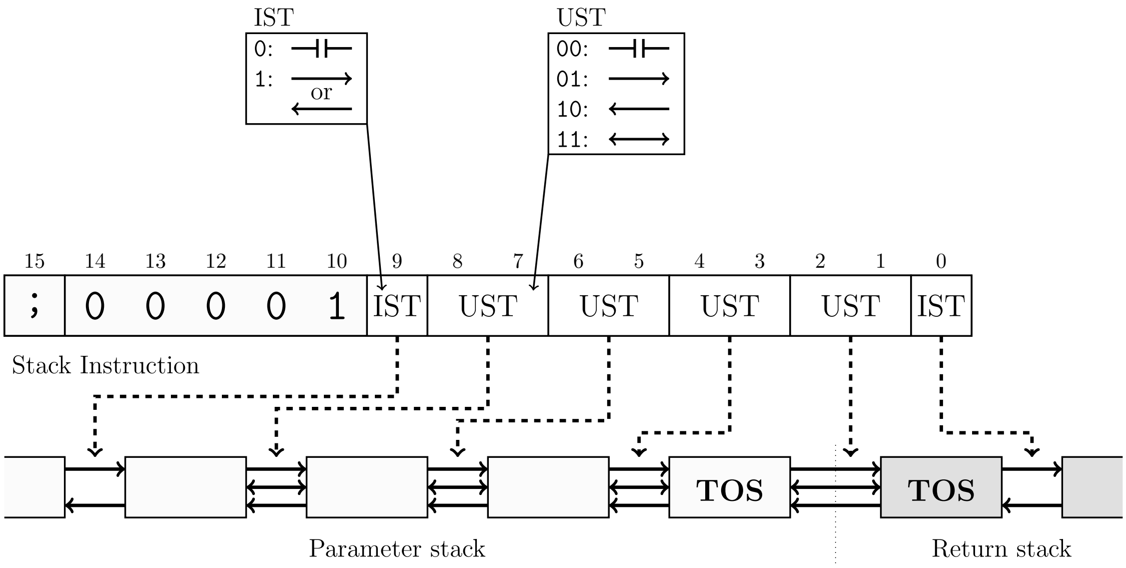


Figure 3-2: Transition encoding of stack instructions

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. 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

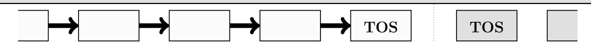
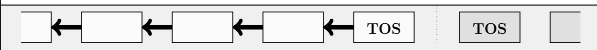
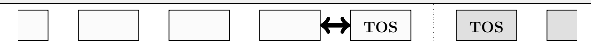
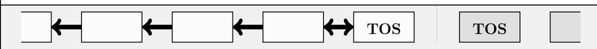
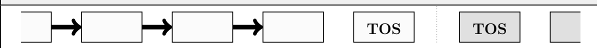
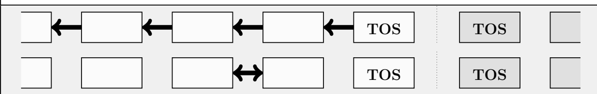
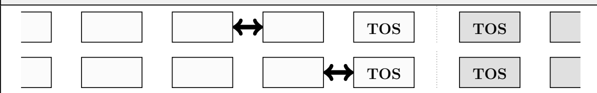
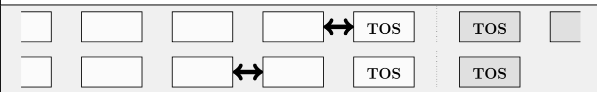
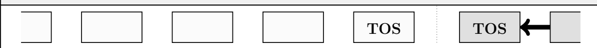
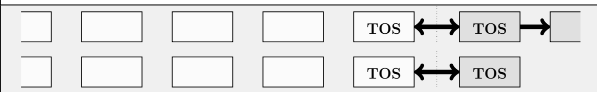
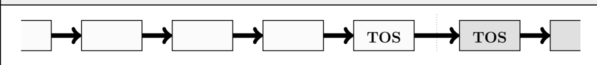
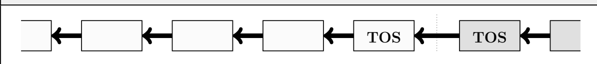
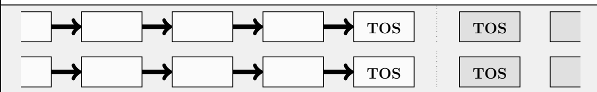
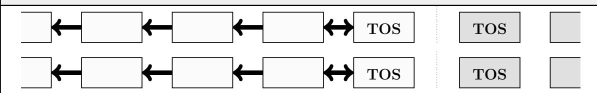
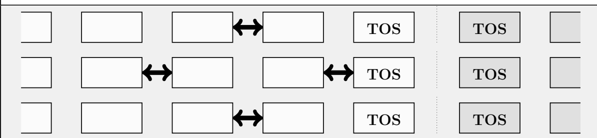
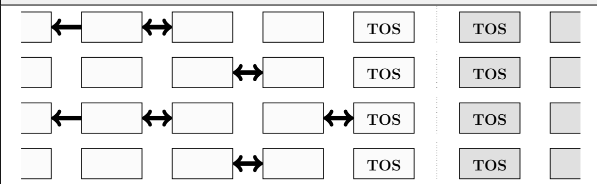
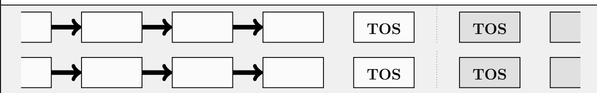
It is possible to put the UST fields into a combination which would trigger a data transfer of two source cells to a single desination cell. In these cases, the resulting data in the desination cell is undefined.

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.

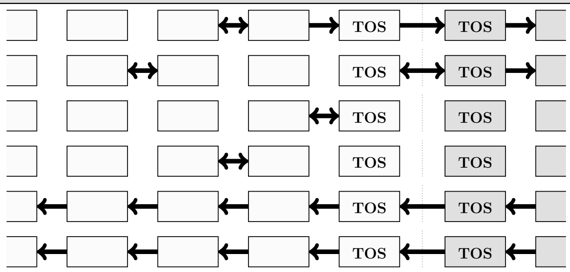
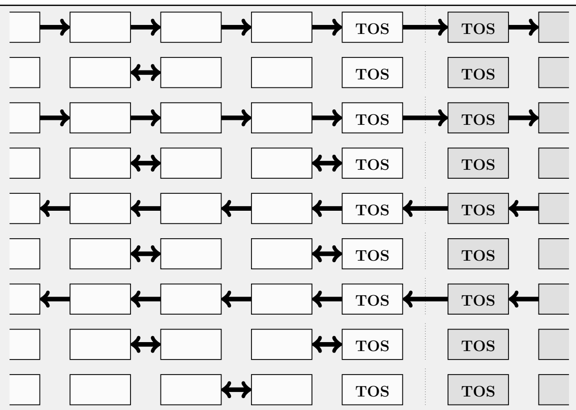
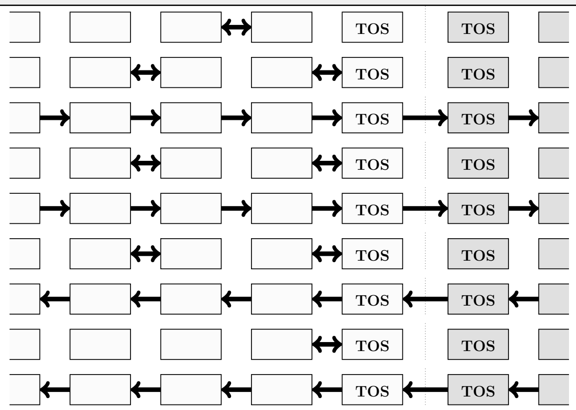
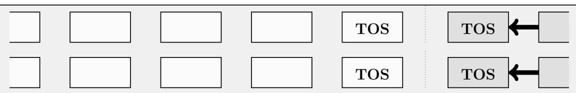
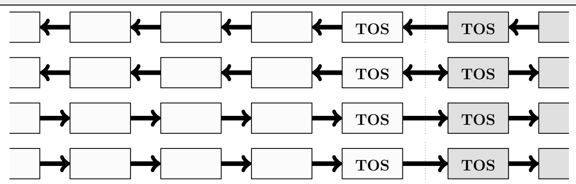
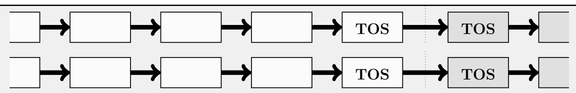
Table 3-2 shows how stack operations in Forth are mapped N1 instructions.

Table 3-2: 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 –)		0x0001
RDUP	(R: x – x x)		0x0007 0x0006
>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

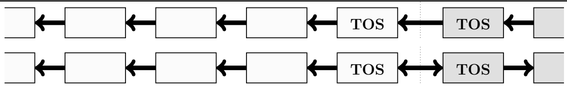
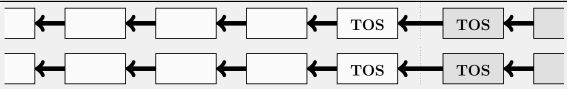
...continued

Table 3-2: Common stack operations

Word	Description	Transitions	Opcode
2TUCK	(x1 x2 x3 x4 – x3 x4 x1 x2 x3 x4)		0x046B 0x0487 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 0x0018 0x0755
2RDROP	(R: x1 x2 –)		0x0001 0x0001
2RDUP	(R: x1 x2 – x1 x1 x1 x2)		0x0755 0x0757 0x06AB 0x06AB
2>R	(x1 x2 –) (R: – x1 x2)		0x06AB 0x06AB

...continued

Table 3-2: 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`

3.8 Memory Access Instructions

Memory access instruction perform read or write acesses to the system's 128KB address space. Data is solely accessed in 16-bit entities. Accesses to a 510B subset of the address space can be done through an immediate addressing. This will offer faster access to frequently used system variables.

3.9 Control Instructions

The N1 implements two types of control instructions to manipulate the internal state of the CPU. The first type are simple control instructions. These don't consume input from the stacks nor do they produce a return value. These instructions can be executed concurrently and combined into a single CPU instruction. Table 3-3 shows the set of simple control instructions and thir encoding.

Table 3-3: Simple Control Instructions

Encoding	Action
`0b0000001xxxxxxx1`	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 3-4 lists the encoding of these complex control instructions.

Table 3-4: Nonconcurrent control instruction encoding

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

...continued

Table 3-4: Nonconcurrent 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 −)

4 ANS Forth Words

Table 4-1 provides a list of standard ANS Forth words (see [1]) which map to hardware instructions of the N1 processor.

Table 4-1: 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-addr –)	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	0x0CF0
`0<>`	(x – flag)	Test if x is not zero	0x0D70
`0>`	(n – flag)	Test if n is greater than zero	0x0CA0
`0=`	(x – flag)	Test if x is not zero	0x0D70
`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 MSB	0x0F41
`2/`	(x1 – x2)	Shift x1 one bit towards the LSB, while the MSB 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)	Drop 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 TOS	0x0750 0x0460 0x0789 0x0460
`2>R`	(x1 x2 –) (R: – x1 x2)	Shift cell pair x1 x2 to the return stack	0x06AB 0x06AB
`2R>`	(– x1 x2) (R: x1 x2 –)	Shift cell pair x1 x2 to the parameter stack	0x0755 0x0755

...continued

Table 4-1: ANS Forth words

Word	Stack	Description	Opcode
2R@	(– x1 x2) (R: x1 x2 – x1 x2)	Copy cell pair x1 x2 to the parameter stack	0x0755 0x0757
2OVER	(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	0x0CE0
<>	(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	0x0CA0
>R	(x –) (R: – x)	Shift x on to the return stack	0x06AB
?DUP	(x – 0|x x)	Duplicate x if it is not zero	0x0750 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 parameter stack without +n	0x00FF
DROP	(x –)	Drop x from the /glsps	0x06A8
DUP	(x – x x)	Duplicate x	0x0750
FALSE	(– false)	FALSE flag	0x1000
INVERT	(x1 – x2)	Bitwise inverse of x1	0x0EBF
LSHIFT	(x1 u – x2)	Shift x1 u bits towards the MSB	0x0F20
M*	(n1 n2 – d)	Multiply n1 by n2	0x0A40
MAX	(n1 n2 – n3)	n3 is the greater of n1 and n2	0x0DB0
MIN	(n1 n2 – n3)	n3 is the lesser of n1 and n2	0x0DF0
NEGATE	(n1 – n2)	n2 is the two's complement of n1	0x0C70
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 TOS	0x0758
R>	(– x) (R: x –)	Shift x to the parameter stack	0x0755
R@	(– x) (R: x – x)	Copy x to the parameter stack	0x0754
RSHIFT	(x1 u – x2)	Shift x1 u bits towards the LSB	0x0F00

...continued

Table 4-1: ANS Forth words

Word	Stack	Description	Opcode
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	0x0CC0
U>	(u1 u2 – flag)	test if u1 is greater than u2	0x0C80
UM*	(u1 u2 – d)	Multiply u1 by u2	0x0A00
XOR	(x1 x2 – x3)	Bitwise logic XOR of x1 and x2	0x0EA0

5 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 5-1, each of these stacks consists of three hardware components: the upper stack, the intermediate stack, and the lower stack.

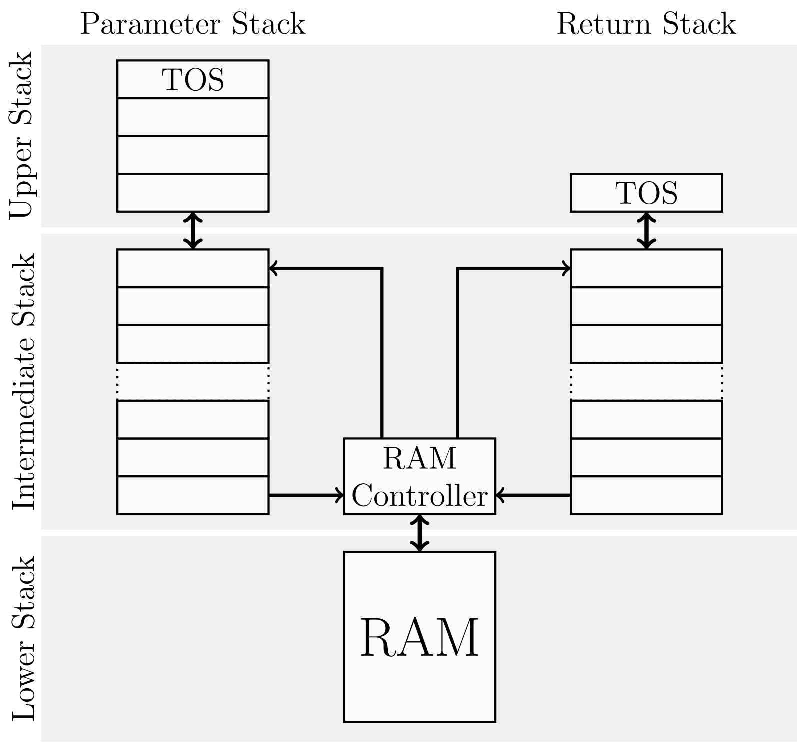


Figure 5-1: Stack Architecture

5.1 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 Instructions" and Section 3.6 "ALU Instructions"). 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 a memory controller that is shared between 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.

5.2 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.

6 Reset, Interrupts, and Exceptions

TBD

7 Integration Guide

TBD

8 Internal Architecture

TBD

9 Verification Status

TBD

10 Tool Summary

TBD

References

[1] American National Standard for Information Systems, 1994.

[2] Menlo Park James Bowman, Willow Garage. J1: a small Forth CPU Core for FPGAs. http://www.excamera.com/files/j1.pdf, 2010.