NSC-1

Nibble Stack Computer 1 (*Preliminary datasheet*)

Abstract
NSC-1 is a 16 bit stack computer special designed for the Forth programming language. It uses 4-bit instuction codes packed within a 16-bit intruction word. With a 4-bit instuction code 16 instructions can be made. This will covers the most used Forth words. The other more hardware complex Forth words are written in small software routines which can be accessed with a fast call. The design fits into small FPGAs and CPLDs. The synthesizible implementation uses VHDL.

Contents

Introduction

In this document the following abbreviations are used:

IW	16 bit Instruction Word	16 bit CPU register
IC	4 bit nibble Instruction Code	4:1 multiplexer
IP	Instruction Pointer	16 bit CPU register
DP	Data stack Pointer	4 bit CPU register
RP	Return stack Pointer	4 bit CPU register
TOS	Top Of data Stack	16 bit CPU register
NOS	Next Of data Stack	
R	top of Return stack	
D-stack	Data stack	17 places
R-stack	Return stack	16 places

Table 1. Abbreviations

NSC-1 has no general purpose registers. All operations are preformed on the D-stack. There is however one extra register which holds top of stack (TOS) because we need simultaneous access to the top two values for some calculations. Therefore the D-stack is 17 place instead of the 16 places of the R-stack. The CAL instruction pushes its return address on the R-stack. There are two instructions to move data from and to D-stack and R-stack: TOR and FMR, to R-stack and from R-stack. Forth words: >R and R>

NSC-1 uses little endianness.

1. Instruction set

Four IC nibbles are packed into one 16 bit IW:

IC4	IC3	IC2	IC1
15..12	11..8	7..4	3..0

Table 2. IC alignment in a IW

NSC-1 available instructions:

Mne	Op	Mne	Op	Mne	Op	Mne	Op
NOP	#0	CPF	#1	ADD	#2	AND	#3
XOR	#4	SAR	#5	DUP	#6	DRP	#7
TOR	#8	FMR	#9	LOD	#A	STO	#B
SLT	#C	LIT	#D	JNZ	#E	CAL	#F

Table 3. NSC-1 Instruction Codes

The intruction set explenation use boxed text for the D-stack contents *before* and *after* the performed intruction.

before → *after*

n1 n2 operation → *result*

n1 = NOS, n2 = TOS

1.1 NOP

No OPeration

The NOP is mainly used as a 16 bit word alignment filler. All jump lables and return addresses has to be aligned to a 16 bit IW.

1.2 CPF

ComPare and load Flags

n1 - n2 → n1 flags

The CPF instruction has a double function. The CPF instruction substracts NOS with TOS and overwrites TOS with the resulting flags. A previous ADD, AND, XOR, SAR instruction registered flag result will also written to TOS.

ZE	ZEro	NZ	Not Zero
SI	SIgn	NS	Not Sign
CY	CarrY	NC	Not Carry
OV	OVerflow	NV	No oVerflow
c	Compare flag	a	Arithmetic flag

Table 4. Flags abbreviations

7	6	5	4	3	2	1	0
cNS	cSI	cNC	cCY	aNC	cNZ	cZE	aCY

Table 5. Flags low byte

15	14	13	12	11	10	9	8
aNV	aOV	aNS	aSI	aNZ	aZE	cNV	cOV

Table 6. Flags high byte

The most used flags are in the first three bits. They can be mask with the short literal SLT instruction. After masking has done then `jnz` condidional jump can be used. If `aCY` is mask after a add then a add with carry can be done.

1.3 ADD

ADDition
Forth word: +

n1 + n2 → sum

Affected flags: aZE, aSI, aCY, aOV

1.4 AND

Logical AND
Forth word: and

n1 ∧ n2 → and

Affected flags: aZE, aSI, aCY

1.5 XOR

Logical XOR
Forth word: xor

n1 ⊕ n2 → xor

Affected flags: aZE, aSI, aCY

1.6 SAR

Shift Arithmetric Rigth

n1 → n1(15) & n1(14..1)

Affected flags: aZE, aSI, aCY

1.7 DUP

DUPlicate
Forth word: dup

n1 → n1 n2

Affected flags: none

1.8 DRP

DRoP
Forth word: drop

n1 n2 → n1

Affected flags: none

1.9 TOR

TO R
Forth word: >R

$\boxed{n1 \rightarrow \wr \rightarrow R}$

Affected flags: none

1.10 FMR

FroM R
Forth word: R>

$\boxed{R \rightarrow \wr \rightarrow n1}$

Affected flags: none

1.11 LOD

LOaD from memory
Forth word: @

$\boxed{addr \rightarrow n1}$

Affected flags: none

LOD translate a TOS values of 0..7 to FFF0..FFF7 (Ref. table 7).

1.12 STO

STOre to memory
Forth word: !

$\boxed{val\ addr \rightarrow}$

Affected flags: none

STO translate a TOS values of 0..7 to FFF0..FFF7 (Ref. table 7).

1.13 SLT

Short LiTeral

$\boxed{\rightarrow n1}$

Affected flags: none

After SLT IC nibble fetch another nibble is fetched, which is the short literal. This short literal is sign extended. The range of the short literal is: -8..7 or 0000..0007 and FFF8..FFFF. With a short literal for example, we can perform a TOS increment or decrement with only three nibble fetches:

```
Exmpl1:         ; Increment TOS
     slt  #1    ; +1
     add

Exmpl2:         ; Decrement TOS
     slt  #F    ; -1
     add
```

The short literal is also useful for 16 fast access memory locations and 16 fast call to subroutines. Therefore a address translation is performed by the LOD, STO and CAL instruction near lower end and upper end of memory.

LOD and STO translate a TOS values of 0000..0007 to FFF0..FFF7 (Ref. table 7). Low memory is ROM and using LOD and STO in this area isn't very useful. Fast memory access example:

```
     slt  #5   ; Variable number to TOS
     lod       ; Load memory variable to TOS

ORG  #FFF5     ; Origin of variable 5
V_5: #1234     ; Define variable 5
```

CAL translate TOS values of 0000..0007 to 0100..0170 and FFF8..FFFF to 0180..01F0 with 16 IW addresses alignment (Ref. table 7). This means a call to the first eight IW can not be done. But 0000 is the reset vector and here begins the startup code. A call to the upper eight memory location is not useful because the upper 16 memory locations is reserved by fast access memory variables. Fast call example:

```
     slt  #1   ; Load routine number
     cal       ; Call routine

ORG  #0110     ; Origin of fast call #1
     ; Max routine size is 16 IW
```

Fast call routines are defined in: `../asm/nsc1.i`

SLT #	TOS hex	TOS dec	LOD/STO	CAL
0	0000	0	FFF0	0100
1	0001	1	FFF1	0110
2	0002	2	FFF2	0120
3	0003	3	FFF3	0130
4	0004	4	FFF4	0140
5	0005	5	FFF5	0150
6	0006	6	FFF6	0160
7	0007	7	FFF7	0170
8	FFF8	-8	FFF8	0180
9	FFF9	-7	FFF9	0190
A	FFFA	-6	FFFA	01A0
B	FFFB	-5	FFFB	01B0
C	FFFC	-4	FFFC	01C0
D	FFFD	-3	FFFD	01D0
E	FFFE	-2	FFFE	01E0
F	FFFF	-1	FFFF	01F0

Table 7. SLT translations

1.14 LIT

LITeral

→ n1

Affected flags: none

After LIT IC nibble fetch another four nibbles are fetched, which is the 16 bit literal. NSC-1 uses little endianness:

nibble 4	nibble 3	nibble 2	nibble 1
15..12	11..8	7..4	3..0

Table 8. LIT fetches

1.15 JNZ

Jump if Not Zero

n1 addr →

Affected flags: none

Jump to addr if n1=0

It is import that the **ALIGN** directive has to be coded before **every** jump label. The **ALIGN** directive takes care of emitting an unfinished IW and filling up the memory with NOP's until the label is 16-bit word aligned.

1.16 CAL

CALl subroutine

R-stack: IP → ≀ → R	D-stack: addr → ≀ → IP

Affected flags: none

Current IW IP address plus one is pushed to R-stack. TOS is popped to IP. All IW nibbles after CAL need to be NOP's.

CAL translate TOS values of 0000..0007 to 0100..0170 and FFF8..FFFF to 0180..01F0 with 16 IW addresses alignment (Ref. table 7).

The folowing code performs a return from a call:

```
fmr   ; Get return address from R
dup   ; NOS is not zero
jnz   ; TOS is jump address
```

2. ISA reference card

To gain assembly code readablility some pseudo instructions are defined in: `../asm/nsc.i`

Table below list all available native and pseudo instructions

Mne	Stack	Description
nop	→	No operation
cpf	n1 - n2 → flags	Compare and load flags
add	n1 + n2 → sum	Addition
and	n1 ∧ n2 → and	Logic AND
xor	n1 ⊕ n2 → and	Logic XOR
sar	n → n(15,15..1)	Arithmetic shift right
dup	n → n n	Duplicate TOS
drp	n →	Drop TOS
tor	n → ≀ → R	Move TOS to R-Stack
fmr	R → ≀ → n	Move R-Stack to TOS
lod	addr → n	Load from memory
sto	n addr →	Store to memory
slt L	→ n	4-bit short literal load
lit L	→ n	16-bit literal load
jnz	flags addr →	Jump if not zero
cal	addr →	Call subroutine on addr
imm L	→ n	Auto use of SLT or LIT
addr A	→ addr	Label address to TOS
call A	→	Call to label
return	→	Return from subroutine
jump A	→	Jump to label
rcpy	→ n	Copy of R to TOS
not	n → $\overline{n}$	Bitwise not
shl	n → n(14..1,0)	Shift TOS left
inc	n → n+1	Increment by one
inc2	n → n+2	Increment by two
dec	n → n-1	decrement by one
dec2	n → n-2	decrement by two
IfNot L, A	→	If TOS≠L then jump
sub	n1 - n2 → diff	Substraction
or	n1 ∨ n2 → or	Logical or
swap	n1 n2 → n2 n1	Swap top two items
over	n1 n2 → n1 n2 n1	Copy of NOS
low	n → n(7..0)	Low byte
high	n → n(15..8)	High byte

Table 9. ISA reference card

Mne	Description
ALIGN	Align to 16-bit word boundry
ORG A	Change code origin
STRING N, S	Define string 'S' with 'N' reserved places

Table 10. NSC-1 directives card

3. Assembler SMAL32

Compiling assembly code is done with the target less compiler SMAL32 by Douglas W. Jones:

http://homepage.divms.uiowa.edu/ jones/cross/smal32/

To get NSC-1 targed by SMAL32 the file `../asm/nsc.i` has to be included in the assembly file. This file contains NSC-1 native and pseudo instructions macro's. plus the fast call routines.