N-105

Programmer's Reference Manual

Rosemary Francis

Computer Laboratory
University of Cambridge

August 2004

Sponsored by:

ALTERA

The Cambridge-MIT Institute

www.cambridge-mit.org

N-105

1 Introduction

The N-105 is a 16-bit embedded soft core processor with a two stage pipeline and a RISC instruction set. It was designed by Rosemary Francis, Daniel Hulme and Simon Moore as a teaching resource. Its is loosely based on Altera's Nios, but contains no derived code. It is designed to be used with Altera's Avalon bus as it has no memory management. It assumes logically separate data and instruction memories and has two independent interfaces to the bus. The two stage pipeline consists of instruction fetch and everything else.

2 Overview

2.1 Registers

There are 16 16-bit general purpose registers and one reserved register called the pc (program counter). The general purpose registers are named r0 to r15. R15 may be read or written to in the same way others are. During a branch to subroutine(BSR) the pc+2 is written to r15 and then r15 is written to pc on a return (RET). You can therefore manually write to the pc using the RET instruction. It is up to the programmer to ensure the contents of r15 is valid during a return. The pc contains the address of the next instruction to execute. It is half word aligned.

2.2 Condition Flags

The condition flags are set by selected alu operations. Some set all flags, others set only a few flags. In the case where only a few are set, the value of those not set will be undefined. If an instruction does not set any flags then they are unaffected and retain the value at which they were last set.

Condition Flags	
N	Negative
V	Arithmetic overflow
Z	Zero
C	Carry for unsigned arithmetic

2.3 Condition codes

There is only one condition instruction : IFS. This allows the instruction following to be executed only if the condition is true. A skipped instruction is still fetched from memory and so there is no time advantage unless the skipped instruction was a load or store.

Condition Codes			
code	value	symbols	meaning
0000	$\overline{C}$	cc nc	Carry not set
0001	C	cs c	Carry set
0010	$\overline{Z}$	ne nz	Not equal, zero not set
0011	Z	eq z	Equal, zero set
0100	$\overline{N}$	p pl	Negative not set, positive
0101	N	n mi	Negitive set
0110	$N \oplus V$	lt	Less than
0111	$\overline{N \oplus V}$	gt	Greater than or equal
0110	$\overline{Z \vee (N \oplus V)}$	gt	Greater than
0111	$Z \vee (N \oplus V)$	le	Less than or equal
0100	$\overline{V}$	vc nv	Overflow not set
0101	V	vs v	Overflow set
0100	$\overline{C \vee Z}$	hi	unsigned higher
0101	$C \vee Z$	la	unsigned lower

2.4 Program Flow

There are three branch instructions: BR, BSR and RET. These all update the pc. However the pc contains the value of the the next instruction to execute which is the instruction currently being fetched. This means the the instruction following a branch will be executed as the pipeline is not flushed. Returns therefore return to the instruction after so that not instructions are executed twice. For example in the code below 8 will be moved to r4 before the pc will point to instruction at three. Also 4 will be moved to r1 before the return takes place. The return sets the pc to the instruction at two. Here nop is a “no op” which is equivalent to “or r0 r0” which does nothing.

```
one    movi r2 4
       bsr  three
       movi r4 8
two    movi r4 6
       movi r9 7
       br end
       nop
three  ret
       movi r1 4
end    nop
```

2.5 Memory Access

Memory access is via Altera's Avalon bus used for their NIOS soft processor. The Avalon bus can handle 16 bit transfers only so addresses must be word aligned. Apart from that there is no memory management. Transfers may only take place when the bus is ready the processor may have to wait indefinitely. Loads and stores typically take only a few clock cycles longer. Instruction fetch works similarly. It typically takes 2 clock cycles to fetch an instruction.

2.6 Instruction Fields

A	Index of source and destination register
B	Index of source register
imm4	4-bit unsigned immediate
imm7	7-bit signed or unsigned immediate
imm11	11-bit signed immediate

Instruction formats

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
A											opcode				
A				B							opcode				
A				imm4							opcode				
A				imm7							opcode				
imm11											opcode				
											opcode				

3 Instruction Set Quick Reference

N-105 Instruction Set

Opcode	Mnemonic	Operands	Function	Flags
000 00	OR	ra rb	ra ← ra | rb	n-z-
000 01	AND	ra rb	ra ← ra & rb	n-z-
000 10	XOR	ra rb	ra ← ra $\oplus$ rb	n-z-
000 11	NOT	ra	ra ← !ra	
001 00	MOV	ra rb	ra ← rb	
001 01	ADD	ra rb	ra ← ra + rb	nvzc
001 10	SUB	ra rb	ra ← ra − rb	nvzc
001 11	CMP	ra rb	ra − rb	nvzc
010 00	LSLI	ra imm4	ra ← ra << 16-imm4	
010 01	LSRI	ra imm4	ra ← ra >>> imm4	
010 10	ASRI	ra imm4	ra ← ra >> imm4	
010 11	ROTI	ra imm4	ra ← ra rotated > imm4	
011 00	MOVI	ra imm7	ra ← imm7	
011 01	ADDI	ra imm7	ra ← ra + imm7	nvzc
011 10	SUBI	ra imm7	ra ← ra − imm7	nvzc
011 11	CMPI	ra imm7	ra − imm7	nvzc
100 00	LD	ra rb	ra ← [rb]	
101 00	ST	ra rb	[rb] ← ra	
110 00	BR	imm11	pc ← pc + imm11	
110 01	BSR	imm11	pc ← pc + imm11	
			r15 ← pc + 2	
110 10	RET		pc ← r15	
111 11	IFS	cc_imm4	skip ← !condition	

ADD

ADD

Operation: rA ← rA + rB

Assembler Syntax: add ra rb

Example: `add r4 r5`

Description: Add the values contained in rA and rB and place the result in rA

Condition codes: Flags set

Instruction Fields: Register index A, register index B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				B				x			0	0	1	0	1

ADDI

ADD immediate

Operation: rA ← rA + imm7

Assembler Syntax: addi rA imm7

Example: `addi r8 3`

Description: Add the values contained in rA and imm7 and place the result in rA

Condition codes: Flags set

Instruction Fields: register index A, unsigned 7-bit immediate imm7

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				imm7							0	1	1	0	1

AND

Bitwise AND

Operation:	rA ← rA & rB
Assembler Syntax:	and rA rB
Example:	`and r14 r0`
Description:	AND the bits in rA and rB and place the result in rA
Condition codes:	Negitive and zero flags set, others set undefined
Instruction Fields:	register index A, register index B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				B				x			0	0	0	0	1

ASRI

Arithmetic right shift

Operation: rA ← rA >> imm4

Assembler Syntax: asri rA imm4

Example: `asri r6 9`

Description: shift rA right by the amount in imm4, shifting in the 15th bit to preserve the sign

Condition codes: Flags unaffected

Instruction Fields: Register index A, 4-bit unsigned immediate imm4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				imm4				x			0	1	0	1	0

BR

Branch

Operation: pc ← pc + imm11

Assembler Syntax: br *addr*

Example: `br loopLabel`

Description: Performs a branch to addr. The delay slot is then executed

Condition codes: Flags unaffected

Instruction Fields: Signed 11-bit immediate imm11

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm11											1	1	0	0	0

BSR

Branch to subroutine

Operation:	pc $\leftarrow$ pc + imm11 r15 $\leftarrow$ pc + 2
Assembler Syntax:	bsr *label*
Example:	`bsr loopLabel`
Description:	Branch to addr and store pc in r15. The delay slot is then executed
Condition codes:	Flags unaffected
Instruction Fields:	Signed 11 bit immediate imm11

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm11											1	1	0	0	1

CMP

Compare

Operation:	rA − rB
Assembler Syntax:	cmp rA rB
Example:	`cmp r9 r3`
Description:	Subtract rB from rA but do not store the result
Condition codes:	Flags set
Instruction Fields:	Register index A, register index B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				B				x			0	0	1	1	1

CMPI

Compare immediate

Operation: rA − imm7

Assembler Syntax: cmpi rA imm7

Example: `cmpi r9 -24`

Description: Subtract imm7 from the contents of rA, but do not store the result

Condition codes: Flags set

Instruction Fields: Register index A, signed 7-bit immediate imm7

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				imm7							0	1	1	1	1

IFS

Conditionally execute next instruction

Operation: skip ← !condition

Assembler Syntax: ifs cc_imm4

Example: `ifs cc_gt`

Description: Skip the next instruction if the condition is false, execute if true

Condition codes: Flags unaffected

Instruction Fields: 4-bit condition immediate imm4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x					imm4				x		1	1	1	1	1

LD

Load

Operation: rA ← [rB]

Assembler Syntax: ld rA rB

Example: `ld r5 r6`

Description: Load the half word at address in rB into rA. Address must be half word aligned

Condition codes: Flags unaffected

Instruction Fields: Register index A, register index B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A											1	0	0	0	0

Bits 15–12: A; bits 11–8: B; bits 7–5: x.

LSLI

Logical left shift

Operation: rA $\leftarrow$ rA $<<$ 16 $-$ imm4

Assembler Syntax: lsli rA 16 $-$ imm4

Example: `lsli r11 13`

Description: Logical left shift by a 4-bit immediate (shifts in 0s)
The four bit immediate is the bottom four bits of 16-imm4

Condition codes: Flags unaffected

Instruction Fields: Register index A, 4-bit immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				imm4				x			0	1	0	0	0

LSRI

Logical right shift

Operation: rA ← rA >> imm4

Assembler Syntax: lsri rA imm4

Example: `lsri r0 5`

Description: Logical shift right by a 4-bit immediate imm4 (shifts in 0s)

Condition codes: Flags unaffected

Instruction Fields: Register index rA, 4-bit immediate imm4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				imm4				x			0	1	0	0	1

MOV

Move

Operation:	rA ← rB
Assembler Syntax:	mov rA rB
Example:	`mov r8 r2`
Description:	Move the contents of rB into rA
Condition codes:	Flags unaffected
Instruction Fields:	Register index rA, register index rB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				B				x			0	0	1	0	0

MOVI

Move immediate

Operation:	rA $\leftarrow$ imm7
Assembler Syntax:	movi rA imm7
Example:	`movi r12 41`
Description:	Move 7-bit signed immediate imm7 into rA (with sign extension)
Condition codes:	Flags unaffected
Instruction Fields:	Register index A, signed 7-bit immediate imm7

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				imm7							0	1	1	0	0

NOP

No operation

Operation:	none
Assembler Syntax:	nop
Example:	`nop`
Description:	Does nothing. Equivalent to “or r0 r0”
Condition codes:	Negitive and zero flags set, others left undefined
Instruction Fields:	none

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

NOT

Bitwise Logical NOT

Operation: rA ← !rA

Assembler Syntax: not rA

Example: `not r3`

Description: Logical not, bitwise inversion

Condition codes: Flags unaffected

Instruction Fields: Register index A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				x							0	0	0	1	1

OR

Bitwise Logical OR

Operation: rA ← rA | rB

Assembler Syntax: or rA rB

Example: `or r9 r2`

Description: Perform bitwise logical or on bits in rA with bits in rB and place in rA

Condition codes: Negative and zero flags set, others left undefined.

Instruction Fields: Register index A, register index B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				B				x			0	0	0	0	0

RET

Return from Subroutine

Operation: pc ← r15

Assembler Syntax: ret

Example: `ret`

Description: Returns from subroutine by writing address saved in r15 to the program counter

Condition codes: Flags unaffected

Instruction Fields: none

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x											1	1	0	1	1

ROTI

Bitwise Rotate Right

Operation: $rA \leftarrow rA >> imm4 \mid ra << 16 - imm4$

Assembler Syntax: roti rA imm4

Example: `roti r9 11`

Description: Rotate the bits in rA right imm4 places

Condition codes: Flags unaffected

Instruction Fields: Register index A, 4-bit unsigned immediate imm4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x					imm4				x		0	1	0	1	1

ST

Store

Operation: [rB] ← rA

Assembler Syntax: st rA rB

Example: `st r15 r8`

Description: Store the value in rA at address in rB. Address must be half word aligned

Condition codes: Flags unaffected

Instruction Fields: Register index A, register index B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				B				x			1	0	1	0	0

SUB

Subtract

Operation:	rA ← rA − rB
Assembler Syntax:	sub rA rB
Example:	`sub r2 r0`
Description:	Subtract rB from rA and store the result back in rA
Condition codes:	Flags set
Instruction Fields:	Register index A, register index B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				B				x			0	0	1	1	0

SUBI

Subtract immediate

Operation: rA ← rA − imm7

Assembler Syntax: subi rA imm7

Example: `subi r9 63`

Description: Subtracts unsigned 7-bit immediate imm7 from rA and stores result back in rA

Condition codes: Flags set

Instruction Fields: Register index rA, unsigned 7-bit immediate imm7

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A				imm7							0	1	1	1	0

XOR

Bitwise Logical Exclusive OR

Operation: rA $\leftarrow$ rA $\oplus$ rB

Assembler Syntax: xor rA rB

Example: `xor r0 r6`

Description: Performs logical bitwise exclusive or on rA and rB and stores the result back in rA

Condition codes: Negative and zero flags set, others left undefined

Instruction Fields: Register index A, register index B