Nios II Instructions

Arithmetic & Logical Instructions

add Rdest, Rsrc1, Rsrc2	signed (with overflow) and unsigned (with carry) addition
addi Rdest, Rsrc1, IMM16	signed (with overflow) and unsigned (with carry) addition immediate
and Rdest, Rsrc1, Rsrc2	AND
andhi Rdest, Rsrc1, IMM16	AND immediate into high halfword
andi Rdest, Rsrc1, IMM16	AND immediate
div Rdest, Rsrc1, Rsrc2	signed divide
divu Rdest, Rsrc1, Rsrc2	unsigned divide
mul Rdest, Rsrc1, Rsrc2	multiply, store the 32 low-order bits of the product to Rdest
muli Rdest, Rsrc1, IMM16	multiply immediate, sign-extend the 16-bit immediate value to 32 bits, store the 32 low-order bits of the product to Rdest
mulxss Rdest, Rsrc1, Rsrc2	signed multiply, store the 32 high-order bits of the product to Rdest
mulxsu Rdest, Rsrc1, Rsrc2	treat Rsrc1 as a signed integer and Rsrc2 as an unsigned interger, store the 32 high-order bits of the product to Rdest
mulxuu Rdest, Rsrc1, Rsrc2	unsigned multiply, store the 32 high-order bits of the product to Rdest
nor Rdest, Rsrc1, Rsrc2	bitwise logical nor
or Rdest, Rsrc1, Rsrc2	bitwise logical or
orhi Rdest, Rsrc1, IMM16	calculate the bitwise logical OR of Rsrc1 and (IMM16:0x0000) and store the result in Rdest
ori Rdest, Rsrc1, IMM16	calculate the bitwise logical OR of Rsrc1 and (0x0000:IMM16) and store the result in Rdest
rol Rdest, Rsrc1, Rsrc2	rotate Rsrc1 left by the number of bits specified in $Rsrc2_{4..0}$, the bits that shift out of the register rotate into the least-significant bit positions
roli Rdest, Rsrc1, IMM5	rotate Rsrc1 left by the number of bits specified in IMM5
ror Rdest, Rsrc1, Rsrc2	rotate Rsrc1 right by the number of bits specified in $Rsrc2_{4..0}$, the bits that shift out of the register rotate into the most-significant bit positions
sll Rdest, Rsrc1, Rsrc2	shift Rsrc1 left by the number of bits specified in $Rsrc2_{4..0}$ (inserting zeros)
slli Rdest, Rsrc1, IMM5	shift Rsrc1 left by the number of bits specified in IMM5 (inserting zeros)
sra Rdest, Rsrc1, Rsrc2	shift Rsrc1 right by the number of bits specified in $Rsrc2_{4..0}$ (duplicating the sign bit)
srai Rdest, Rsrc1, IMM5	shift Rsrc1 right by the number of bits specified in IMM5 (duplicating the sign bit)
srl Rdest, Rsrc1, Rsrc2	shift Rsrc1 right by the number of bits specified in $Rsrc2_{4..0}$ (inserting zeros)
srli Rdest, Rsrc1, IMM5	shift Rsrc1 right by the number of bits specified in IMM5 (inserting zeros)
sub Rdest, Rsrc1, Rsrc2	signed (with overflow) and unsigned (with carry) subtraction
subi Rdest, Rsrc1, IMM16	signed (with overflow) and unsigned (with carry) subtraction immediate
xor Rdest, Rsrc1, Rsrc2	calculate the bitwise logical exclusive XOR of Rsrc1 and Rsrc2
xorhi Rdest, Rsrc1, IMM16	calculate the bitwise logical exclusive XOR of Rsrc1 and (IMM16:0x0000)
xori Rdest, Rsrc1, IMM16	calculate the bitwise logical exclusive XOR of Rsrc1 and (0x0000:IMM16)

Comparison Instructions

cmpeq Rdest, Rsrc1, Rsrc2	compare equal, Rdest = 1 if Rsrc1 == Rsrc2; otherwise Rdest = 0
cmpeqi Rdest, Rsrc1, IMM16	sign-extend the 16-bit immediate value IMM16 to 32 bits and compare it to the value of Rsrc1, if equal, Rdest = 1; otherwise Rdest = 0
cmpge Rdest, Rsrc1, Rsrc2	signed compare, if Rsrc1 >= Rsrc2, Rdest = 1; otherwise Rdest = 0
cmpgei Rdest, Rsrc1, IMM16	sign-extend the 16-bit immediate value IMM16 to 32 bits and compare it to the value of Rsrc1, if Rsrc1 >= IMM16, Rdest = 1; otherwise Rdest = 0
cmpgeu Rdest, Rsrc1, Rsrc2	unsigned compare, if Rsrc1 >= Rsrc2, Rdest = 1; otherwise Rdest = 0
cmpgeui Rdest, Rsrc1, IMM16	zero-extend the 16-bit immediate value IMM16 to 32 bits and compare it to the value of Rsrc1, if Rsrc1 >= IMM16, Rdest = 1; otherwise Rdest = 0
cmpgt Rdest, Rsrc1, Rsrc2	signed compare, if Rsrc1 > Rsrc2, Rdest = 1; otherwise Rdest = 0
cmpgti Rdest, Rsrc1, IMMED	sign-extend the 16-bit immediate value IMMED to 32 bits and compare it to the value of Rsrc1, if Rsrc1 > IMMED, Rdest = 1; otherwise Rdest = 0
cmpgtu Rdest, Rsrc1, Rsrc2	unsigned compare, if Rsrc1 > Rsrc2, Rdest = 1; otherwise Rdest = 0

cmpgtui Rdest, Rsrc1, IMMED	zero-extend the 16-bit immediate value IMMED to 32 bits and compare it to the value of Rsrc1, if Rsrc1 > IMMED, Rdest = 1; otherwise Rdest = 0
cmple Rdest, Rsrc1, Rsrc2	signed compare, if Rsrc1 <= Rsrc2, Rdest = 1; otherwise Rdest = 0
cmplei Rdest, Rsrc1, IMMED	sign-extend the 16-bit immediate value IMMED to 32 bits and compare it to the value of Rsrc1, if Rsrc1 <= IMMED, Rdest = 1; otherwise Rdest = 0
cmpleu Rdest, Rsrc1, Rsrc2	unsigned compare, if Rsrc1 <= Rsrc2, Rdest = 1; otherwise Rdest = 0
cmpleui Rdest, Rsrc1, IMMED	zero-extend the 16-bit immediate value IMMED to 32 bits and compare it to the value of Rsrc1, if Rsrc1 <= IMMED, Rdest = 1; otherwise Rdest = 0
cmplt Rdest, Rsrc1, Rsrc2	signed compare, if Rsrc1 < Rsrc2, Rdest = 1; otherwise Rdest = 0
cmplti Rdest, Rsrc1, IMM16	sign-extend the 16-bit immediate value IMMED to 32 bits and compare it to the value of Rsrc1, if Rsrc1 < IMM16, Rdest = 1; otherwise Rdest = 0
cmpltu Rdest, Rsrc1, Rsrc2	unsigned compare, if Rsrc1 < Rsrc2, Rdest = 1; otherwise Rdest = 0
cmpltui Rdest, Rsrc1, IMM16	zero-extend the 16-bit immediate value IMMED to 32 bits and compare it to the value of Rsrc1, if Rsrc1 < IMM16, Rdest = 1; otherwise Rdest = 0
cmpne Rdest, Rsrc1, Rsrc2	compare not equal, Rdest = 1 if Rsrc1 == Rsrc2; otherwise Rdest = 0
cmpnei Rdest, Rsrc1, IMM16	sign-extend the 16-bit immediate value IMM16 to 32 bits and compare it to the value of Rsrc1, if not equal, Rdest = 1; otherwise Rdest = 0

Branch and Jump Instructions

beq Rsrc1, Rsrc2, label	branch if equal
bge Rsrc1, Rsrc2, label	signed branch if Rsrc1 greater than or equal to Rsrc2
bgeu Rsrc1, Rsrc2, label	unsigned branch if Rsrc1 greater than or equal to Rsrc2
bgt Rsrc1, Rsrc2, label	signed branch if Rsrc1 greater than Rsrc2
bgtu Rsrc1, Rsrc2, label	unsigned branch if Rsrc1 greater than Rsrc2
ble Rsrc1, Rsrc2, label	signed branch if Rsrc1 less than or equal to Rsrc2
bleu Rsrc1, Rsrc2, label	unsigned branch if Rsrc1 less than or equal to Rsrc2
blt Rsrc1, Rsrc2, label	signed branch if Rsrc1 less than Rsrc2
bltu Rsrc1, Rsrc2, label	unsigned branch if Rsrc1 less than Rsrc2
bne Rssrc1, Rsrc2, label	branch if not equal
br label	unconditional branch
break	debugging breakpoint
bret	breakpoint return
call label	call subroutine
callr Rsrc1	call subroutine in register, the value in Rsrc1 is the address of the next instruction
eret	exception return
jump Rsrc1	transfer execution to the address contained in Rsrc1
ret	return from subroutine

Load Instructions

Load byte from memory or I/O peripheral

ldb/ldbio Rdest, byte_offset(Rsrc1)	compute the effective byte address specified by the sum of Rsrc1 and byte_offset, load the byte into Rdest and sigh-extend the 8-bit value to 32 bits
ldbu/ldbuio Rdest, byte_offset(Rsrc1)	compute the effective byte address specified by the sum of Rsrc1 and byte_offset, load the byte into Rdest and zero-extend the 8-bit value to 32 bits

Load half word from memory or I/O peripheral

ldh/ldhio Rdest, byte_offset(Rsrc1)	compute the effective byte address specified by the sum of Rsrc1 and byte_offset, load the half word into Rdest and sign-extend the 16-bit value to 32 bits
ldhu/ldhuio Rdest, byte_offset(Rsrc1)	compute the effective byte address specified by the sum of Rsrc1 and byte_offset, load the half word into Rdest and zero-extend the 16-bit value to 32 bits

Load word from memory or I/O peripheral

ldw/ldwio Rdest, byte_offset(Rsrc1)	compute the effective byte address specified by the sum of Rsrc1 and byte_offset, load the word into Rdest

Store Instructions

Store byte to memory or I/O peripheral

stb/stbio Rsrc1, byte_offset(Rsrc2)	compute the effective byte address specified by the sum of Rsrc1 and byte_offset, store the low byte to the memory byte specified by the effective address

Store half word from memory or I/O peripheral

sth/sthio Rdest, byte_offset(Rsrc1)	compute the effective byte address specified by the sum of Rsrc1 and byte_offset, store the low halfword to the memory location specified by the effective address

Store word from memory or I/O peripheral

stw/stwio Rdest, byte_offset(Rsrc1)	compute the effective byte address specified by the sum of Rsrc1 and byte_offset, store the word to the memory location specified by the effective address

Data Movement Instructions

mov Rdest, Rsrc1	move register to register
movhi Rdest, IMMED	move immediate into high halfword, and clear the lower halfword of Rdest to 0x0000
movi Rdest, IMMED	move signed immedtiate into word
movia Rdest, label	move immediate address into word
movui Rdest, IMMED	move unsigned immediate into word, and zero-extend the immediate value IMMED to 32 bits

Others

nextpc Rdest	store the address of the next instruction to Rdest
nop	no operation
rdctl Rdest, ctlN	read from control register
wrctl ctlN, Rsrc1	write to control register