The NiMi-iSC 16 bit Instruction-Set architecture

Revision 1

NiMi-iSC 16-bit Instruction Set overview

This instruction set is meant to be a middle ground between overly generalized architectures such as RiSC-16[1] and overly specific architectures such as x86. NiMiiSC-16 is an 8-register, 16-bit computer. The word size is 16 bits and all buses. All addresses are word-addresses (i.e., address 0 corresponds to the first two bytes of main memory, address 1 corresponds to the second two bytes of main memory, etc.). In each instruction 5 bits are dedicated to the opcode. Big endian will be used throughout the instruction set.

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

Formats:

RRR	Opcode	Reg a	Reg b	Reg c	0
RI	Opcode	Reg a	Immediate		
D	Opcode	Data			
RR	Opcode	Reg a	Reg b	0	
R	Opcode	Reg a	0		
N	Opcode	0			

Instructions

ADD	Opcode	Reg a	Reg b	Reg c	0
SUB	Opcode	Reg a	Reg b	Reg c	0
ADDI	Opcode	Reg a	Data		
SUBI	Opcode	Reg a	Data		
SUBII	Opcode	Reg a	Data		
OR	Opcode	Reg a	Reg b	Reg c	0
AND	Opcode	Reg a	Reg b	Reg c	0

[1] (Jacob)

NOT	Opcode	Reg a	Reg b	0	
XOR	Opcode	Reg a	Reg b	Reg c	0
EQ	Opcode	Reg a	Reg b	Reg c	0
NEQ	Opcode	Reg a	Reg b	Reg c	0
LESS	Opcode	Reg a	Reg b	Reg c	0
LOE	Opcode	Reg a	Reg b	Reg c	0
GRE	Opcode	Reg a	Reg b	Reg c	0
GOE	Opcode	Reg a	Reg b	Reg c	0
STR	Opcode	Reg a	Reg b	0	
LDR	Opcode	Reg a	Reg b	0	
PUSH	Opcode	Reg a	0		
POP	Opcode	Reg a	0		
CALL	Opcode	Addr.			
RET	Opcode	0			
BITL	Opcode	Reg a	Reg b	Reg c	0
BITR	Opcode	Reg a	Reg b	Reg c	0
HTL	Opcode	0			
IMM	Opcode	Data			
MOV	Opcode	Reg a	Reg b	0	

The following table describes the different instruction operators

Mnemonic	Name and format	Opcode (Binary)	Assembly format	Description
ADD	Add RRR-type		add rA, rB, rC	Add rA with rB and store in rC
SUB	Subtract RRR-type		sub rA, rB, rC	Subtract rB from rA and store in rC
ADDI	Add Immediate RI-type		addi rA, Imm	Add rA with Imm, store in reg 3

SUBI	Subtract Immediate RI-type		subi rA, Imm	Subtract Imm from rA
SUBII	Subtract Immediate 2 RI-type		subii rA, Imm	Subtract rA from Imm.
OR	Or RRR-type		or rA, rB, rC	Or the contents of rA with rB, store result in rC.
AND	AND RRR-type		and rA, rB, rC	And the contents of rA with rB, store result in rC.
NOT	NOT RRR-type		not rA, rB	Not the contents of rA, store result in rB.
XOR	Exclusive OR RRR-type		xor rA, rB, rC	Xor the contents of rA with rB, store result in rC.
EQU	Equal RRR-type		eq rA, rB, rC	If the contents of rA and rB are the same, jump to the address stored in rC.
NEQ	Not Equal RRR-type		neq rA, rB, rC	If the contents of rA and rB are not the same, jump to the address stored in rC.
LES	Less RRR-type		les rA, rB, rC	If the contents of rA are less than rB, jump to the address stored in rC.
LOE	Less Or Equal RRR-type		loe rA, rB, rC	If the contents of rA are less or equal to rB, jump to the address stored in rC.
GRE	Greater RRR-type		gre rA, rB, rC	If the contents of rA are greater than rB, jump to the address stored in rC
GOE	Greater Or Equal RRR-type		goe rA, rB, rC	If the contents of rA are greater or equal to rB, jump to the address stored in rC
STR	Store RR-type		str rA, rB	Store contents of rA in the ram address located at rB
LDR	Load RR-type		ldr rA, rB	Load contents into rA from the ram address located at rB

PUSH	Push R-type		push rA	Push contents of rA onto the stack
POP	Pop R-type		pop rA	Pop contents of stack onto rA
CALL	Call D-type		call imm	Call function with address at
RET	Return N-type		ret	Return from function call
BITL	Bitwise left RRR-type		bitl rA, rB, rC	Bitwise left rA by rB bits (max 16), store in rC
BITR	Bitwise right RRR-type		bitr rA, rB, rC	Bitwise right rA by rB bits (max 16), store in rC
HLT	Halt clock N-type		hlt	Stop the clock
IMM	Immediate D-type		imm imm	Immediate imm into reg 0
MOV	Move RR-type		mov rA, rB	Move contents of rA into rB

Registers

0) Reg0 *
1) Reg1
2) Reg2
3) Reg3 **
4) Reg4
5) Reg5
6) Counter
7) I/O ***

* Reg0 Stores the Immediate value when imm instruction is used

** Reg3 stores the output when immediate calculate functions are used (e.g., ADDI)

*** I/O register will receive user input when it is placed in an input position, and it will output to user when put in a output register position.