Abacus Language Reference Card

conventions and registers

bv2nat(b)	interpret bitvector b as radix-2 number
bv2int(b)	interpret bitvector b as two's-complement number
Reg[r]	scalar register r (r$\in \{0,\ldots,7\}$)
Vec[r]	vector register r (r$\in \{0,\ldots,7\}$)
ovflw	register for double precision arithmetic
pc	program counter
ll	load-linked register
vlen	vector length register
vmask	vector mask register

load/store instructions

scalar load/store instructions

ld rd,rl,rr	load word	Reg[rd] = Mem[**bv2nat**(Reg[rl])+ **bv2nat**(Reg[rr])]
st rd,rl,rr	store word	Mem[**bv2nat**(Reg[rl])+ **bv2nat**(Reg[rr])] = Reg[rd]
ldi rd,rl,c	load word immed.	Reg[rd] = Mem[**bv2nat**(Reg[rl])+ **bv2nat**(c)]
sti rd,rl,c	store word immed.	Mem[**bv2nat**(Reg[rl])+ **bv2nat**(c)] = Reg[rd]

vector load/store instructions

lv vd,rl,rr	load vector	**for**(i=0..15)Vec[vd][i] = Mem[**bv2nat**(Reg[rl])+**bv2nat**(Reg[rr])+i]
sv vd,rl,rr	store vector	**for**(i=0..15)Mem[**bv2nat**(Reg[rl])+**bv2nat**(Reg[rr])+i] = Vec[vd][i]
lvws vd,rl,rr	load vec. w. stride	**for**(i=0..15)Vec[vd][i] = Mem[**bv2nat**(Reg[rl])+i***bv2nat**(Reg[rr])]
svws vd,rl,rr	store vec. w. stride	**for**(i=0..15)Mem[**bv2nat**(Reg[rl])+i***bv2nat**(Reg[rr])] = Vec[vd][i]

register move instructions

mov rd,c	move signed constant to register	Reg[rd] = **bv2int**(c)
ovf rd	move ovflw to register	Reg[rd] = **ovflw**
mvtm rd	move register to mask register	**vmask** = Reg[rd]
mvtl rd	move register to vector length register	**vlen** = Reg[rd]

thread synchronization

ll rd,rl,c	load linked	Reg[rd] = Mem[**bv2nat**(Reg[rl])+**bv2nat**(c)] **ll** = bv2nat(Reg[rl])+bv2nat(c)
sc rd,rl,c	store conditional	Mem[**bv2nat**(Reg[rl])+**bv2nat**(c)] = Reg[rd] Reg[rd] = **ll**
sync	write back dirty cache blocks	

branch/jump instructions

bez rd,c	branch if zero	**if**(Reg[rd]==0)pc = pc + **bv2int**(c)
bnz rd,c	branch if not zero	**if**(Reg[rd]!=0)pc = pc + **bv2int**(c)
jmp rd	jump to register address	pc = pc + **bv2int**(Reg[rd])
j c	jump to immediate address	pc = pc + **bv2int**(c)

pseudo instructions

nop	do nothing ⇝ **j** 1
movr rd,rl	register transfer ⇝ **addiu** rd,rl,0
mod rd,rl,rr	remainder (signed) ⇝ **div** rd,rl,rr, **ovf** rd
modu rd,rl,rr	remainder (unsigned) ⇝ **divu** rd,rl,rr, **ovf** rd
modi rd,rl,c	remainder immediate (signed) ⇝ **divi** rd,rl,rr, **ovf** rd
modiu rd,rl,c	remainder immediate(unsigned) ⇝ **diviu** rd,rl,rr, **ovf** rd
sgt rd,rl,rr	set greater than (signed) ⇝ **slt** rd,rr,rl
sgtu rd,rl,rr	set greater than (unsigned) ⇝ **sltu** rd,rr,rl
sge rd,rl,rr	set gt. than/eq. to (signed) ⇝ **sle** rd,rr,rl
sgeu rd,rl,rr	set gt. than/eq. to (unsigned) ⇝ **sleu** rd,rr,rl
vsgt rd,vl,vr	set greater than (signed) ⇝ **vslt** rd,vr,vl
vsgtu rd,vl,vr	set greater than (unsigned) ⇝ **vsltu** rd,vr,vl
vsge rd,vl,vr	set gt. than/eq. to (signed) ⇝ **vsle** rd,vr,vl
vsgeu rd,vl,vr	set gt. than/eq. to (unsigned) ⇝ **vsleu** rd,vr,vl
blt rd,rl,rr,c	branch on less-than (signed) ⇝ **slt** rd,rl,rr; **bnz** rd,c
bltu rd,rl,rr,c	branch on less-than (unsigned) ⇝ **sltu** rd,rl,rr; **bnz** rd,c
ble rd,rl,rr,c	branch on less-equal (signed) ⇝ **sle** rd,rl,rr; **bnz** rd,c
bleu rd,rl,rr,c	branch on less-equal (unsigned) ⇝ **sleu** rd,rl,rr; **bnz** rd,c
bgt rd,rl,rr,c	branch on greater-than (signed) ⇝ **slt** rd,rr,rl; **bnz** rd,c
bgtu rd,rl,rr,c	branch on greater-than (unsigned) ⇝ **sltu** rd,rr,rl; **bnz** rd,c
bge rd,rl,rr,c	branch on greater-equal (signed) ⇝ **sle** rd,rr,rl; **bnz** rd,c
bgeu rd,rl,rr,c	branch on greater-equal (unsigned)⇝ **sleu** rd,rr,rl; **bnz** rd,c

scalar instructions

scalar arithmetic/logic instructions

add rd,rl,rr	add signed	Reg[rd] = **bv2int**(Reg[rl])+ **bv2int**(Reg[rr])
addu rd,rl,rr	add unsigned	Reg[rd] = **bv2nat**(Reg[rl])+ **bv2nat**(Reg[rr])
addi rd,rl,c	add signed immediate	Reg[rd] = **bv2int**(Reg[rl])+ **bv2int**(c)
addiu rd,rl,c	add unsigned immediate	Reg[rd] = **bv2nat**(Reg[rl])+ **bv2nat**(c)
sub rd,rl,rr	subtract signed	Reg[rd] = **bv2int**(Reg[rl])- **bv2int**(Reg[rr])
subu rd,rl,rr	subtract unsigned	Reg[rd] = **bv2nat**(Reg[rl])- **bv2nat**(Reg[rr])
subi rd,rl,c	subtract signed immediate	Reg[rd] = **bv2int**(Reg[rl])- **bv2int**(c)
subiu rd,rl,c	subtract unsigned immediate	Reg[rd] = **bv2nat**(Reg[rl])- **bv2nat**(c)
mul rd,rl,rr	multiply signed	Reg[rd] = **bv2int**(Reg[rl])* **bv2int**(Reg[rr])
mulu rd,rl,rr	multiply unsigned	Reg[rd] = **bv2nat**(Reg[rl])* **bv2nat**(Reg[rr])
muli rd,rl,c	multiply signed immediate	Reg[rd] = **bv2int**(Reg[rl])* **bv2int**(c)
muliu rd,rl,c	multiply unsigned immediate	Reg[rd] = **bv2nat**(Reg[rl])* **bv2nat**(c)
div rd,rl,rr	divide signed	Reg[rd] = **bv2int**(Reg[rl])/ **bv2int**(Reg[rr])
divu rd,rl,rr	divide unsigned	Reg[rd] = **bv2nat**(Reg[rl])/ **bv2nat**(Reg[rr])
divi rd,rl,c	divide signed immediate	Reg[rd] = **bv2int**(Reg[rl])/ **bv2int**(c)
diviu rd,rl,c	divide unsigned immediate	Reg[rd] = **bv2nat**(Reg[rl])/ **bv2nat**(c)

scalar comparison operations

slt rd,rl,rr	set less-than	Reg[rd] = **bv2int**(Reg[rl])< **bv2int**(Reg[rr])
sltu rd,rl,rr	set less-than unsigned	Reg[rd] = **bv2nat**(Reg[rl])< **bv2nat**(Reg[rr])
sle rd,rl,rr	set less-than-equal	Reg[rd] = **bv2int**(Reg[rl])<= **bv2int**(Reg[rr])
sleu rd,rl,rr	set less-than-equal unsigned	Reg[rd] = **bv2nat**(Reg[rl])<= **bv2nat**(Reg[rr])
seq rd,rl,rr	set equal	Reg[rd] = Reg[rl] == Reg[rr]
sne rd,rl,rr	set not equal	Reg[rd] = Reg[rl] != Reg[rr]

scalar bitwise logic operations

and rd,rl,rr	bitwise and	Reg[rd] = Reg[rl] & Reg[rr]
or rd,rl,rr	bitwise or	Reg[rd] = Reg[rl] | Reg[rr]
xor rd,rl,rr	bitwise exclusive or	Reg[rd] = Reg[rl] **xor** Reg[rr]
neg rd,rl	bitwise negation	Reg[rd] = !Reg[rl]
sft rd,rl,rr	left shift of Reg[rl] by Reg[rr] bits into Reg[rd]	

vector instructions

vector arithmetic/logic instructions

vadd vd,vl,vr	add signed	Vec[vd] = **bv2int**(Vec[vl])+ **bv2int**(Vec[vr])
vaddu vd,vl,vr	add unsigned	Vec[vd] = **bv2nat**(Vec[vl])+ **bv2nat**(Vec[vr])
vsub vd,vl,vr	subtract signed	Vec[vd] = **bv2int**(Vec[vl])- **bv2int**(Vec[vr])
vsubu vd,vl,vr	subtract unsigned	Vec[vd] = **bv2nat**(Vec[vl])- **bv2nat**(Vec[vr])
vmul vd,vl,vr	multiply signed	Vec[vd] = **bv2int**(Vec[vl])* **bv2int**(Vec[vr])
vmulu vd,vl,vr	multiply unsigned	Vec[vd] = **bv2nat**(Vec[vl])* **bv2nat**(Vec[vr])
vdiv vd,vl,vr	divide signed	Vec[vd] = **bv2int**(Vec[vl])/ **bv2int**(Vec[vr])
vdivu vd,vl,vr	divide unsigned	Vec[vd] = **bv2nat**(Vec[vl])/ **bv2nat**(Vec[vr])

vector comparison operations

vslt rd,vl,vr	set less-than	Reg[rd] = **bv2int**(Vec[vl])< **bv2int**(Vec[vr])
vsltu rd,vl,vr	set less-than unsigned	Reg[rd] = **bv2nat**(Vec[vl])< **bv2nat**(Vec[vr])
vsle rd,vl,vr	set less-than-equal	Reg[rd] = **bv2int**(Vec[vl])<= **bv2int**(Vec[vr])
vsleu rd,vl,vr	set less-than-equal unsigned	Reg[rd] = **bv2nat**(Vec[vl])<= **bv2nat**(Vec[vr])
vseq rd,vl,vr	set equal	Reg[rd] = Vec[vl] == Vec[vr]
vsne rd,vl,vr	set not equal	Reg[rd] = Vec[vl] != Vec[vr]

vector bitwise logic operations

vand vd,vl,vr	bitwise and	Vec[vd] = Vec[vl] & Vec[vr]
vor vd,vl,vr	bitwise or	Vec[vd] = Vec[vl] | Vec[vr]
vxor vd,vl,vr	bitwise exclusive-or	Vec[vd] = Vec[vl] **xor** Vec[vr]
vneg vd,vl	bitwise negation	Vec[vd] = !Vec[vl]
vsft vd,vl,rr	left shift of Vec[vl] by Reg[rr] bits into Vec[vd]	