Hexagon V60 HVX Instruction Quick Reference

Hexagon™ processor register operands

Symbol	Description
Vs, Vu, Vv	Vector source (512- or 1024-bit)
Vd	Vector destination
Vx	Vector source/destination
Vuu, Vvv	Vector pair source (1024- or 2048-bit)
Vdd	Vector pair destination
Vxx	Vector pair source/destination
Rs, Rt, Ru	Register source (32-bit)
Rd	Register destination
Rx	Register source/destination
Pv	Predicate register
Mu	Modify register
OsN.new	Result value from operation in same packet
Qs, Qt, Qu, Qv	Vector predicate register source (512-bit)
Qd	Vector predicate register destination
Qx	Vector predicate register source/destination

Constant operands

Symbol	Description
#u*N*	*N*-bit unsigned value
#s*N*	*N*-bit signed value

Slot/resource/latency summary

		core slot usage				HVX resources						
Insn	variation	3	2	1	0	ld	mpy	mpy	shift	xlane	st	Early Sources
ALU	no R; 1*vec	any					any					
	no R; 2*vec	any					either pair					
	Rt; 1*vec	either					either					
	Rtt		■				■	■				
Abs-diff	1*vec	either					either					vu/vv
	2*vec	either					■	■				vu/vv
Multiply	by 8b; 1*vec	either					either					vu/vv
	by 8b; 2*vec	either					■	■				vu/vv
	by 16b	either					■	■				vu/vv
Cross-lane	1*vec	any								■		vu/vv
	2*vec	any							■	■		vu/vv or (vx,vy)
Shift or count	1*vec	any							■			vu/vv
load	aligned			either		■	any					
	aligned; .tmp			either		■						
	aligned; .cur			either		■	any					
	unaligned			■	■	■				■		
store	aligned				■		any				■	
	aligned; .new				■						■	
	unaligned			■	■					■	■	vs
histogram		any				■	■	■	■	■	■	
extract		■	■	■	■	■	■	■	■	■	■	~+15

HVX

HVX/ALU-DOUBLE-RESOURCE

Predicate operations
Qd4=and(Qs4,[!]Qt4)
Qd4=or(Qs4,[!]Qt4)
Qd4=xor(Qs4,Qt4)

Combine
Vdd=vcombine(Vu,Vv)
if ([!]Ps) Vdd=vcombine(Vu,Vv)

In-lane shuffle
Vdd.b=vshuffoe(Vu.b,Vv.b)
Vdd.h=vshuffoe(Vu.h,Vv.h)

Swap
Vdd=vswap(Qt4,Vu,Vv)

Sign/Zero extension
Vdd.h=vsxt(Vu.b)
Vdd.uh=vzxt(Vu.ub)
Vdd.uw=vzxt(Vu.uh)
Vdd.w=vsxt(Vu.h)
Vdd=vsxtb(Vu)
Vdd=vsxth(Vu)
Vdd=vzxtb(Vu)
Vdd=vzxth(Vu)

Arithmetic
Vdd.b=vadd(Vuu.b,Vvv.b)
Vdd.b=vsub(Vuu.b,Vvv.b)
Vdd.h=vadd(Vuu.h,Vvv.h)[:sat]
Vdd.h=vsub(Vuu.h,Vvv.h)[:sat]
Vdd.ub=vadd(Vuu.ub,Vvv.ub):sat
Vdd.ub=vsub(Vuu.ub,Vvv.ub):sat
Vdd.uh=vadd(Vuu.uh,Vvv.uh):sat
Vdd.uh=vsub(Vuu.uh,Vvv.uh):sat
Vdd.w=vadd(Vuu.w,Vvv.w)[:sat]
Vdd.w=vsub(Vuu.w,Vvv.w)[:sat]

HVX/ALU-RESOURCE

Predicate operations
Qd4=not(Qs4)

Min/max
Vd.h=vmax(Vu.h,Vv.h)
Vd.h=vmin(Vu.h,Vv.h)
Vd.ub=vmax(Vu.ub,Vv.ub)
Vd.ub=vmin(Vu.ub,Vv.ub)
Vd.uh=vmax(Vu.uh,Vv.uh)
Vd.uh=vmin(Vu.uh,Vv.uh)
Vd.w=vmax(Vu.w,Vv.w)
Vd.w=vmin(Vu.w,Vv.w)

Absolute value
Vd.h=vabs(Vu.h)[:sat]
Vd.w=vabs(Vu.w)[:sat]

Arithmetic
Vd.b=vadd(Vu.b,Vv.b)
Vd.b=vsub(Vu.b,Vv.b)
Vd.h=vadd(Vu.h,Vv.h)[:sat]
Vd.h=vsub(Vu.h,Vv.h)[:sat]
Vd.ub=vadd(Vu.ub,Vv.ub):sat
Vd.ub=vsub(Vu.ub,Vv.ub):sat
Vd.uh=vadd(Vu.uh,Vv.uh):sat
Vd.uh=vsub(Vu.uh,Vv.uh):sat
Vd.w=vadd(Vu.w,Vv.w)[:sat]
Vd.w=vsub(Vu.w,Vv.w)[:sat]

Logical operators
Vd=vand(Vu,Vv)
Vd=vnot(Vu)
Vd=vor(Vu,Vv)
Vd=vxor(Vu,Vv)

Copy
Vd=Vu
if ([!]Ps) Vd=Vu

Average
Vd.b=vnavg(Vu.ub,Vv.ub)
Vd.h=vavg(Vu.h,Vv.h)[:rnd]
Vd.h=vnavg(Vu.h,Vv.h)
Vd.ub=vavg(Vu.ub,Vv.ub)[:rnd]
Vd.uh=vavg(Vu.uh,Vv.uh)[:rnd]
Vd.w=vavg(Vu.w,Vv.w)[:rnd]
Vd.w=vnavg(Vu.w,Vv.w)

Compare vectors
Qd4=vcmp.eq(Vu.b,Vv.b)
Qd4=vcmp.eq(Vu.h,Vv.h)
Qd4=vcmp.eq(Vu.ub,Vv.ub)
Qd4=vcmp.eq(Vu.uh,Vv.uh)
Qd4=vcmp.eq(Vu.uw,Vv.uw)
Qd4=vcmp.eq(Vu.w,Vv.w)
Qd4=vcmp.gt(Vu.b,Vv.b)
Qd4=vcmp.gt(Vu.h,Vv.h)
Qd4=vcmp.gt(Vu.ub,Vv.ub)
Qd4=vcmp.gt(Vu.uh,Vv.uh)
Qd4=vcmp.gt(Vu.uw,Vv.uw)
Qd4=vcmp.gt(Vu.w,Vv.w)
Qx4[&|]=vcmp.eq(Vu.b,Vv.b)
Qx4[&|]=vcmp.eq(Vu.h,Vv.h)
Qx4[&|]=vcmp.eq(Vu.ub,Vv.ub)
Qx4[&|]=vcmp.eq(Vu.uh,Vv.uh)
Qx4[&|]=vcmp.eq(Vu.uw,Vv.uw)
Qx4[&|]=vcmp.eq(Vu.w,Vv.w)
Qx4[&|]=vcmp.gt(Vu.b,Vv.b)
Qx4[&|]=vcmp.gt(Vu.h,Vv.h)
Qx4[&|]=vcmp.gt(Vu.ub,Vv.ub)
Qx4[&|]=vcmp.gt(Vu.uh,Vv.uh)
Qx4[&|]=vcmp.gt(Vu.uw,Vv.uw)
Qx4[&|]=vcmp.gt(Vu.w,Vv.w)
Qx4^=vcmp.eq(Vu.b,Vv.b)
Qx4^=vcmp.eq(Vu.h,Vv.h)
Qx4^=vcmp.eq(Vu.ub,Vv.ub)
Qx4^=vcmp.eq(Vu.uh,Vv.uh)
Qx4^=vcmp.eq(Vu.uw,Vv.uw)
Qx4^=vcmp.eq(Vu.w,Vv.w)
Qx4^=vcmp.gt(Vu.b,Vv.b)
Qx4^=vcmp.gt(Vu.h,Vv.h)
Qx4^=vcmp.gt(Vu.ub,Vv.ub)
Qx4^=vcmp.gt(Vu.uh,Vv.uh)
Qx4^=vcmp.gt(Vu.uw,Vv.uw)
Qx4^=vcmp.gt(Vu.w,Vv.w)

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Conditional accumulate
if ([!]Qv4) Vx.b[+-]=Vu.b
if ([!]Qv4) Vx.h[+-]=Vu.h
if ([!]Qv4) Vx.w[+-]=Vu.w

Mux select
Vd=vmux(Qt4,Vu,Vv)

Saturation
Vd.h=vsat(Vu.w,Vv.w)
Vd.ub=vsat(Vu.h,Vv.h)

In-lane shuffle
Vd.b=vshuffe(Vu.b,Vv.b)
Vd.b=vshuffo(Vu.b,Vv.b)
Vd.h=vshuffe(Vu.h,Vv.h)
Vd.h=vshuffo(Vu.h,Vv.h)

HVX/DEBUG

Extract vector element
Rd.w=vextract(Vu,Rs)
Rd=vextract(Vu,Rs)

HVX/LOAD

Load - aligned
Vd=vmem(Rt)
Vd=vmem(Rt):nt
Vd=vmem(Rt+#s4)
Vd=vmem(Rt+#s4):nt
Vd=vmem(Rx++#s3)
Vd=vmem(Rx++#s3):nt
Vd=vmem(Rx++Mu)
Vd=vmem(Rx++Mu):nt

Load - immediate use
Vd.cur=vmem(Rt+#s4)
Vd.cur=vmem(Rt+#s4):nt
Vd.cur=vmem(Rx++#s3)
Vd.cur=vmem(Rx++#s3):nt
Vd.cur=vmem(Rx++Mu)
Vd.cur=vmem(Rx++Mu):nt

Load - temporary immediate use
Vd.tmp=vmem(Rt+#s4)
Vd.tmp=vmem(Rt+#s4):nt
Vd.tmp=vmem(Rx++#s3)
Vd.tmp=vmem(Rx++#s3):nt
Vd.tmp=vmem(Rx++Mu)
Vd.tmp=vmem(Rx++Mu):nt

Load - unaligned
Vd=vmemu(Rt)
Vd=vmemu(Rt+#s4)
Vd=vmemu(Rx++#s3)
Vd=vmemu(Rx++Mu)

HVX/MPY-DOUBLE-RESOURCE

Arithmetic widening
Vdd.h=vadd(Vu.ub,Vv.ub)
Vdd.h=vsub(Vu.ub,Vv.ub)
Vdd.w=vadd(Vu.h,Vv.h)
Vdd.w=vadd(Vu.uh,Vv.uh)
Vdd.w=vsub(Vu.h,Vv.h)
Vdd.w=vsub(Vu.uh,Vv.uh)

Multiply with 2-wide reduction
Vd.w=vdmpy(Vu.h,Rt.h):sat
Vd.w=vdmpy(Vu.h,Rt.uh):sat
Vd.w=vdmpy(Vu.h,Vv.h):sat
Vd.w=vdmpy(Vuu.h,Rt.h):sat
Vd.w=vdmpy(Vuu.h,Rt.uh,#1):sat
Vdd.h=vdmpy(Vuu.ub,Rt.b)
Vdd.w=vdmpy(Vuu.h,Rt.b)
Vx.w+=vdmpy(Vu.h,Rt.h):sat
Vx.w+=vdmpy(Vu.h,Rt.uh):sat
Vx.w+=vdmpy(Vu.h,Vv.h):sat
Vx.w+=vdmpy(Vuu.h,Rt.h):sat
Vx.w+=vdmpy(Vuu.h,Rt.uh,#1):sat
Vxx.h+=vdmpy(Vuu.ub,Rt.b)
Vxx.w+=vdmpy(Vuu.h,Rt.b)

Multiply-add
Vdd.h=vmpa(Vuu.ub,Rt.b)
Vdd.h=vmpa(Vuu.ub,Vvv.b)
Vdd.h=vmpa(Vuu.ub,Vvv.ub)
Vdd.w=vmpa(Vuu.h,Rt.b)
Vxx.h+=vmpa(Vuu.ub,Rt.b)
Vxx.w+=vmpa(Vuu.h,Rt.b)

Multiply - vector by scalar
Vd.h=vmpy(Vu.h,Rt.h):<<1:rnd:sat
Vd.h=vmpy(Vu.h,Rt.h):<<1:sat
Vdd.h=vmpy(Vu.ub,Rt.b)
Vdd.uh=vmpy(Vu.ub,Rt.ub)
Vdd.uw=vmpy(Vu.uh,Rt.uh)
Vdd.w=vmpy(Vu.h,Rt.h)
Vxx.h+=vmpy(Vu.ub,Rt.b)
Vxx.uh+=vmpy(Vu.ub,Rt.ub)
Vxx.uw+=vmpy(Vu.uh,Rt.uh)
Vxx.w+=vmpy(Vu.h,Rt.h):sat

Multiply - vector by vector
Vd.h=vmpy(Vu.h,Vv.h):<<1:rnd:sat
Vdd.h=vmpy(Vu.b,Vv.b)
Vdd.h=vmpy(Vu.ub,Vv.b)
Vdd.uh=vmpy(Vu.ub,Vv.ub)
Vdd.uw=vmpy(Vu.uh,Vv.uh)
Vdd.w=vmpy(Vu.h,Vv.h)
Vdd.w=vmpy(Vu.h,Vv.uh)
Vxx.h+=vmpy(Vu.b,Vv.b)
Vxx.h+=vmpy(Vu.ub,Vv.b)
Vxx.uh+=vmpy(Vu.ub,Vv.ub)
Vxx.uw+=vmpy(Vu.uh,Vv.uh)
Vxx.w+=vmpy(Vu.h,Vv.h)
Vxx.w+=vmpy(Vu.h,Vv.uh)

Integer multiply - vector by vector
Vd.h=vmpyi(Vu.h,Vv.h)
Vx.h+=vmpyi(Vu.h,Vv.h)

Integer Multiply (32x16)
Vd.w=vmpyie(Vu.w,Vv.uh)
Vd.w=vmpyio(Vu.w,Vv.h)
Vx.w+=vmpyie(Vu.w,Vv.h)
Vx.w+=vmpyie(Vu.w,Vv.uh)

Integer multiply accumulate even/odd
Vd.w=vmpyi(Vu.w,Rt.h)
Vx.w+=vmpyi(Vu.w,Rt.h)

Multiply (32x16)
Vd.w=vmpye(Vu.w,Vv.uh)
Vd.w=vmpyo(Vu.w,Vv.h):<<1[:rnd]:sat
Vx.w+=vmpyo(Vu.w,Vv.h):<<1[:rnd]:sat:shift

Multiply bytes with 4-wide reduction - vector by scalar
Vdd.uw=vrmpy(Vuu.ub,Rt.ub,#u1)
Vdd.w=vrmpy(Vuu.ub,Rt.b,#u1)
Vxx.uw+=vrmpy(Vuu.ub,Rt.ub,#u1)
Vxx.w+=vrmpy(Vuu.ub,Rt.b,#u1)

Multiply accumulate with 4-wide reduction - vector by vector
Vx.uw+=vrmpy(Vu.ub,Vv.ub)
Vx.w+=vrmpy(Vu.b,Vv.b)
Vx.w+=vrmpy(Vu.ub,Vv.b)

Multiply with 3-wide reduction
Vdd.h=vtmpy(Vuu.b,Rt.b)
Vdd.h=vtmpy(Vuu.ub,Rt.b)
Vdd.w=vtmpy(Vuu.h,Rt.b)
Vxx.h+=vtmpy(Vuu.b,Rt.b)
Vxx.h+=vtmpy(Vuu.ub,Rt.b)
Vxx.w+=vtmpy(Vuu.h,Rt.b)

Sum of reduction of absolute differences halfwords
Vdd.uw=vdsad(Vuu.uh,Rt.uh)
Vxx.uw+=vdsad(Vuu.uh,Rt.uh)

Sum of absolute differences byte
Vdd.uw=vrsad(Vuu.ub,Rt.ub,#u1)
Vxx.uw+=vrsad(Vuu.ub,Rt.ub,#u1)

HVX/MPY-RESOURCE

Multiply with 2-wide reduction
Vd.h=vdmpy(Vu.ub,Rt.b)
Vd.w=vdmpy(Vu.h,Rt.b)
Vx.h+=vdmpy(Vu.ub,Rt.b)
Vx.w+=vdmpy(Vu.h,Rt.b)

Integer multiply - even by odd
Vd.w=vmpyieo(Vu.h,Vv.h)

Integer multiply even/odd
Vd.h=vmpyi(Vu.h,Rt.b)
Vd.w=vmpyi(Vu.w,Rt.b)
Vx.h+=vmpyi(Vu.h,Rt.b)
Vx.w+=vmpyi(Vu.w,Rt.b)

Multiply bytes with 4-wide reduction - vector by scalar
Vd.uw=vrmpy(Vu.ub,Rt.ub)
Vd.w=vrmpy(Vu.ub,Rt.b)
Vx.uw+=vrmpy(Vu.ub,Rt.ub)
Vx.w+=vrmpy(Vu.ub,Rt.b)

Multiply with 4-wide reduction - vector by vector
Vd.uw=vrmpy(Vu.ub,Vv.ub)
Vd.w=vrmpy(Vu.b,Vv.b)
Vd.w=vrmpy(Vu.ub,Vv.b)

Splat word from scalar
Vd=vsplat(Rt)

Vector to predicate transfer
Qd4=vand(Vu,Rt)
Qx4|=vand(Vu,Rt)

Predicate to vector transfer
Vd=vand(Qu4,Rt)
Vx|=vand(Qu4,Rt)

Absolute value of difference
Vd.ub=vabsdiff(Vu.ub,Vv.ub)
Vd.uh=vabsdiff(Vu.h,Vv.h)
Vd.uh=vabsdiff(Vu.uh,Vv.uh)
Vd.uw=vabsdiff(Vu.w,Vv.w)

Insert element
Vx.w=vinsert(Rt)

HVX/MULTICYCLE

Histogram
vhist
vhist(Qv4)

HVX/PERMUTE-RESOURCE

Byte alignment
Vd=valign(Vu,Vv,#u3)
Vd=valign(Vu,Vv,Rt)
Vd=vlalign(Vu,Vv,#u3)
Vd=vlalign(Vu,Vv,Rt)
Vd=vror(Vu,Rt)

General permute network
Vd=vdelta(Vu,Vv)
Vd=vrdelta(Vu,Vv)

Shuffle - Deal
Vd.b=vdeal(Vu.b)
Vd.b=vdeale(Vu.b,Vv.b)
Vd.b=vshuff(Vu.b)
Vd.h=vdeal(Vu.h)
Vd.h=vshuff(Vu.h)

Vector in-lane lookup table
Vd.b=vlut32(Vu.b,Vv.b,Rt)

Pack
Vd.b=vpack(Vu.h,Vv.h):sat
Vd.b=vpacke(Vu.h,Vv.h)
Vd.b=vpacko(Vu.h,Vv.h)
Vd.h=vpack(Vu.w,Vv.w):sat
Vd.h=vpacke(Vu.w,Vv.w)
Vd.h=vpacko(Vu.w,Vv.w)
Vd.ub=vpack(Vu.h,Vv.h):sat
Vd.uh=vpack(Vu.w,Vv.w):sat

Set predicate
Qd4=vsetq(Rt)

HVX/PERMUTE-SHIFT-RESOURCE

Vector in-lane lookup table
Vdd.h=vlut16(Vu.b,Vv.h,Rt)
Vx.b|=vlut32(Vu.b,Vv.b,Rt)
Vxx.h|=vlut16(Vu.b,Vv.h,Rt)

Vector shuffle and deal cross-lane
Vdd=vdeal(Vu,Vv,Rt)
Vdd=vshuff(Vu,Vv,Rt)
vdeal(Vy,Vx,Rt)
vshuff(Vy,Vx,Rt)
vtrans2x2(Vy,Vx,Rt)

Unpack
Vdd.h=vunpack(Vu.b)
Vdd.uh=vunpack(Vu.ub)
Vdd.uw=vunpack(Vu.uh)
Vdd.w=vunpack(Vu.h)
Vxx.h|=vunpacko(Vu.b)
Vxx.w|=vunpacko(Vu.h)

HVX/SHIFT-RESOURCE

Narrowing Shift
Vd.b=vasr(Vu.h,Vv.h,Rt):rnd:sat
Vd.h=vasr(Vu.w,Vv.w,Rt):rnd:sat
Vd.h=vasr(Vu.w,Vv.w,Rt)[:sat]
Vd.ub=vasr(Vu.h,Vv.h,Rt)[:rnd]:sat
Vd.uh=vasr(Vu.w,Vv.w,Rt):sat

Shift and add
Vx.w+=vasl(Vu.w,Rt)
Vx.w+=vasr(Vu.w,Rt)

Shift
Vd.b=vasr(Vu.h,Vv.h,Rt):rnd:sat
Vd.h=vasl(Vu.h,Rt)
Vd.h=vasl(Vu.h,Vv.h)
Vd.h=vasr(Vu.h,Rt)
Vd.h=vasr(Vu.h,Vv.h)
Vd.h=vasr(Vu.w,Vv.w,Rt):rnd:sat
Vd.h=vasr(Vu.w,Vv.w,Rt)[:sat]
Vd.h=vlsr(Vu.h,Vv.h)
Vd.ub=vasr(Vu.h,Vv.h,Rt)[:rnd]:sat
Vd.uh=vasr(Vu.w,Vv.w,Rt):sat
Vd.uh=vlsr(Vu.uh,Rt)
Vd.uw=vlsr(Vu.uw,Rt)
Vd.w=vasl(Vu.w,Rt)
Vd.w=vasl(Vu.w,Vv.w)
Vd.w=vasr(Vu.w,Rt)
Vd.w=vasr(Vu.w,Vv.w)
Vd.w=vlsr(Vu.w,Vv.w)

Round to next smaller element size
Vd.b=vround(Vu.h,Vv.h):sat
Vd.h=vround(Vu.w,Vv.w):sat
Vd.ub=vround(Vu.h,Vv.h):sat
Vd.uh=vround(Vu.w,Vv.w):sat

Bit counting
Vd.h=vnormamt(Vu.h)
Vd.h=vpopcount(Vu.h)
Vd.uh=vcl0(Vu.uh)
Vd.uw=vcl0(Vu.uw)
Vd.w=vnormamt(Vu.w)

HVX/STORE

Store - byte-enabled aligned
if ([!]Qv4) vmem(Rt):nt=Vs
if ([!]Qv4) vmem(Rt)=Vs
if ([!]Qv4) vmem(Rt+#s4):nt=Vs
if ([!]Qv4) vmem(Rt+#s4)=Vs
if ([!]Qv4) vmem(Rx++#s3):nt=Vs
if ([!]Qv4) vmem(Rx++#s3)=Vs
if ([!]Qv4) vmem(Rx++Mu):nt=Vs
if ([!]Qv4) vmem(Rx++Mu)=Vs

Store - new
if ([!]Pv) vmem(Rt+#s4):nt=Os8.new
if ([!]Pv) vmem(Rt+#s4)=Os8.new
if ([!]Pv) vmem(Rx++#s3):nt=Os8.new
if ([!]Pv) vmem(Rx++#s3)=Os8.new
if ([!]Pv) vmem(Rx++Mu):nt=Os8.new
if ([!]Pv) vmem(Rx++Mu)=Os8.new
vmem(Rt):nt=Os8.new
vmem(Rt)=Os8.new
vmem(Rt+#s4):nt=Os8.new
vmem(Rt+#s4)=Os8.new
vmem(Rx++#s3):nt=Os8.new
vmem(Rx++#s3)=Os8.new
vmem(Rx++Mu):nt=Os8.new
vmem(Rx++Mu)=Os8.new

Store - aligned
if ([!]Pv) vmem(Rt):nt=Vs
if ([!]Pv) vmem(Rt)=Vs
if ([!]Pv) vmem(Rt+#s4):nt=Vs
if ([!]Pv) vmem(Rt+#s4)=Vs
if ([!]Pv) vmem(Rx++#s3):nt=Vs
if ([!]Pv) vmem(Rx++#s3)=Vs
if ([!]Pv) vmem(Rx++Mu):nt=Vs
if ([!]Pv) vmem(Rx++Mu)=Vs
vmem(Rt):nt=Vs
vmem(Rt)=Vs
vmem(Rt+#s4):nt=Vs
vmem(Rt+#s4)=Vs
vmem(Rx++#s3):nt=Vs
vmem(Rx++#s3)=Vs
vmem(Rx++Mu):nt=Vs
vmem(Rx++Mu)=Vs

Store - unaligned
if ([!]Pv) vmemu(Rt)=Vs
if ([!]Pv) vmemu(Rt+#s4)=Vs
if ([!]Pv) vmemu(Rx++#s3)=Vs
if ([!]Pv) vmemu(Rx++Mu)=Vs
vmemu(Rt)=Vs
vmemu(Rt+#s4)=Vs
vmemu(Rx++#s3)=Vs
vmemu(Rx++Mu)=Vs