PPU Instruction Set

Table 1: Version control

Version	Updated	Description
1.2	Yoav 6/24/2005	First Draft
1.3	Yoav 7/8/2005	(1) Small change in the AGU instruction move the update flag from the control register into the instruction. (2) Add operand to the shuffle network instructions
1.4	Yoav 8/9/2005	(1) Add immediate trf instruction (2) Add Vector move instruction
1.5	Yoav 8/18/2005	(1) integrate the ALU mode into the instruction (2) Permutation supported only as a distinct instruction (3) Add select register (4) Update AGU

1.Instruction Operation and Execution notations

Table 2: Symbol Definitions

Symbol	Meaning
Vd	Vector destination register
Sd	Scalar destination register
Ad	Address destination register
Pd	Predication register destination
Vn	Vector source 1
Sn	Scalar source 1
An	Address source 1
(An.s)	address pointer 1 points to scalar[a]
(An.v)	address pointer 1 points to vector
Pn	Predication register source 1
PSR	Scalar status bit
Vm.s	address pointer 2 points to scalar
Vm.v	address pointer 2 points to vector
Sm	Scalar source 2
Am	AGU source 2
(Am.s)	address pointer 2 points to scalar
(Am.v)	address pointer 2 points to vector
Pm	Predication register source 2
#imm	Immediate value
Rd	Vd or Sd
Rn	Vn or Sn
Rm	Vm or Sm or imm
L	Loop counter
+	Increment AGU as define in the AGU control register

Table 2: Symbol Definitions

Symbol	Meaning
-	Decrement AGU as define in the AGU control register

a. The pointer can be to the Scalar memory or to Vector memory depends on the address

2. Instruction Partition

The PPU supports a concurrent execution of two units, the address generator unit (AGU) and the Data units (DALU). This enable to execute on the data path while reading from the memory. Both AGU and DALU can be either scalar, vector or vector scalar cross instruction.

Table 3: Instruction packing

Part A	Part B
Data Instruction	AGU Instruction

Table 4: Function unit partition

DALU					AGU
Arithmetic	Logic	Data Control	Vector Scalar	Vector Permutation	move
nop neg add addu addc sub subu subc mul mulu mulc mull mullu mullc mac * macu * macc * div (pseudo) mod(pseudo)	and or xor not lsl lsr asr not	cmpne cmpeq cmpgt cmplt cmple bt bf jmp select trf trfq trfs	dot min max	vcs shup shupw shdn shdnw hfup hfdn bfly shrk expd	nop move.b move.w move.l move.2l move.q move.v

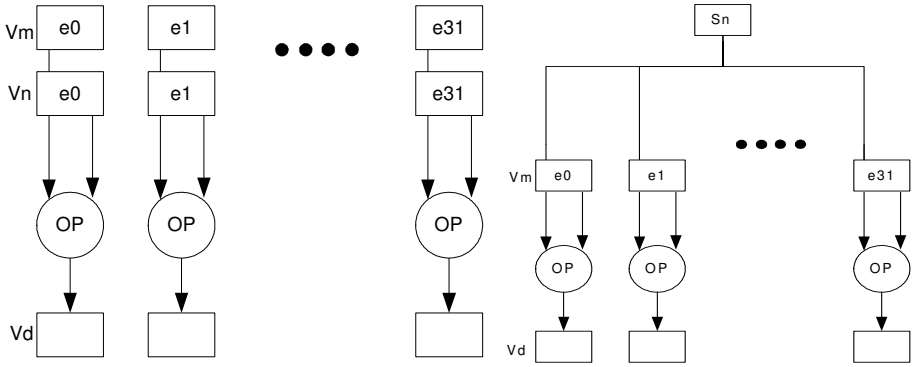
3. Arithmetic and Logic instructions

3.0 The Arithmetic and Logic commands

The arithmetic/logic instruction includes a predication annotation, two operands and destination.
CMD <perd> dst,src1,src2

The PPU supports three types of operands: Vector to Vector, Vector and Scalar to Vector and Scalar to Scalar.

1. Vd,Vm,Vn - Vector to Vector operation
2. Vd,Vm,Sn/Imm - Scalar or Immediate to Vector operation
3. Sd,Sm,Sn - Scalar to Scalar



3.1 Carry

In order to support precision higher than 8bit, both the ALU and the Multiplier include register to hold the carry and logic to add the carry to the input operands. The following table summarize this operation.

Table 5: Carry operation on the Multiplier

Instruction	Carry L	Carry H	Out
add/sub	Overflow	N/A	LSB
addc/subc	Overflow	N/A	LSB+L
mul	-	MSB	LSB
mulc	LSB+H	MSB	LSB+H
mull	-	-	LSB
mullc	-	MSB+H	LSB+L

3.2 Overflow/Saturation

The arithmetic instruction can be executed in two mode (Overflow) and (Saturation).
The execution mode controlled by the MODE register.

3.3 Predication

The predication option enable to control the operation with one of the four bit vectors p0,p1,p2,p3 called predicators and operates as follow:

<pred_cond,pn>

(1) <true,Pn> - Execute on true condition

(2) <false,Pn> - Execute on false condition

(3) <value> - Execute on (p3:p0 == Value)

(4) <inv:Pn Pm> - The signed of the operands defined by the predicator

3.4 Instruction list

Table 6: Arithmetic and Logic Instruction list

Symbol	Name	Operands	flags
nop	No operation	-	-
neg	Negative	Rd,Rn,Rm	<pred>
add	Add	Rd,Rn,Rm	<pred>
addc	Add with carry	Rd,Rn,Rm	<pred>
addu	Unsigned add	Rd,Rn,Rm	<pred>
sub	Subtract	Rd,Rn,Rm	<pred>
subc	Subtract with carry	Rd,Rn,Rm	<pred>
subu	Unsigned subtract	Rd,Rn,Rm	<pred>
mul	Multiply	Rd,Rn,Rm	<pred>
mulc	Multiply with carry	Rd,Rn,Rm	<pred>
mulu	Unsigned multiplication	Rd,Rn,Rm	<pred>
mull	Multiply low	Rd,Rn,Rm	<pred>
mullc	Multiply low with carry	Rd,Rn,Rm	<pred>

Table 6: Arithmetic and Logic Instruction list

Symbol	Name	Operands	flags
mullu	Multiply low unsigned	Rd,Rn,Rm	<pred>
mac	Multiply and accumulate	Rd,Rn,Rm	<pred>
div	Divided	Rd,Rn,Rm	<pred>
mod		Rd,Rn,Rm	<pred>
and	And	Rd,Rn,Rm	<pred>
or	Or	Rd,Rn,Rm	<pred>
xor	Xor	Rd,Rn,Rm	<pred>
not	Not	Rd,Rn,Rm	<pred>
lsl	Logic shift left	Sd,Sm/IMM Vd,Sm/IMM	<pred>
lsr	Logic shift right	Sd,Sm/IMM Vd,Sm/IMM	<pred>
asl	Arithmetic shift left	Sd,Sm/IMM Vd,Sm/IMM	<pred>

4. Data control

4.0 Compare commands

Compare two operands and write to predicator or status register.

4.1 Compare commands operands

The PPU supports three types of operands (1) compare vectors and update the predicator (2) compare vector with scalar and update the predicator (3) Compare two scalar and update the PSR

1. Pd,Vm,Vn
2. Pd,Vm,Sn/Imm
3. Sm,Sn

4.2 Select

The select instruction select between two operands, according to the predicator

4.3 Flow commands (bt,bf,jmp)
The flow instructions are only supported in the scalar unit

CMD <Lable/Sm>

4.4 Transfer instruction

The transfer instruction moves data between registers. The following transfer instruction are supported:

Table 7: Transfer instruction types

Instruction	Description
trf	Transfer data between registers
trfq	Transfer data between Scalar and Vector using a queue.
trfs	Transfer data between Scalar/Imm to special purpose register specify by it address.

Table 8: Transfer Instructions operands

Operands	Description
trf Vd,Vm	Vector register to vector register transfer
trf Vd,Sm/Imm	8-LSB of scalar register duplicate and transfer to vector
trf Vd,Sm,Sn	8-LSB of scalar to a Vector element index by Sn
trf Vd,Pm	Bit vector transfer to Vector
trf Sd,Sm/Imm	Transfer a scalar or immediate to scalar
trf Sd,Vm,Sn/Imm	Transfer an indexed element (Sn/Imm) for vector Vm to scalar Sd.
trf Sd,Am	Transfer an AGU register to Scalar
trf.l Sd,Pm	Transfer low portion of the predicator to Scalar
trf.h Sd,Pm	Transfer high portion of the predicator to Scalar
trf Pd,Sm:Sn	Transfer two sequential registers to predicator
trf Pd,Vm	Transfer a vector LSB into predicator

Table 8: Transfer Instructions operands

Operands	Description
trf.l Pd,Vm	Transfer 4LSB of vector to 4 predicators
trf.h Pd,Vm	Transfer 4MSB of the vector to 4 predicators
trf Ad,Sm/Imm	Transfer Scalar register or immediate to AGU register
trfq Vd,Sm:Sn	Transfer Sm:Sn to queue after 8 transaction to the queue a complete vector will be written to Vd
trfs SPR[#ADD],Sm/Imm trfs Sm,SPR[#ADD]	Transfer a scalar register to a special purpose register Transfer a special purpose register to a scalar

Table 9: Data control Instruction list

Symbol	Name	Operands
cmpne	Compare not equal	Pd,Vm,Vn Pd,Vm,Sn/Imm Sm,Sn
cmpeq	Compare equal	Pd,Vm,Vn Pd,Vm,Sn/Imm Sm,Sn
cmpgt	Compare greater than	Pd,Vm,Vn Pd,Vm,Sn/Imm Sm,Sn
cmplt	Compare less then	Pd,Vm,Vn Pd,Vm,Sn/Imm Sm,Sn
cmple	Compare less equal	Pd,Vm,Vn Pd,Vm,Sn/Imm Sm,Sn
bt	Branch true	Sd/IMM
bf	Branch false	Sd/IMM
jmp	Jump	Sd/IMM

Table 9: Data control Instruction list

Symbol	Name	Operands
select	Select	Rd,Rm,Rn
trf<.l,.h>	Transfer	Vd,Vm/Sm/Imm/Sm,Sn/Pm Sd,Sm/Vm,Sn/Am/Pm Pd,Sm:Sn,Vm,Pm Ad,Sm/IMM
trfq	Transfer to queue	Vd,Sm:Sn
trfs	Transfer to special purpose register	SPR[#Add],Sm Sd,SPR[#Add]

5. Vector to Scalar

dot - Sum all vector elements with 16bit scalar and write the results to scalar
min - Calculate the min value and the min index of all vector elements and a scalar, put the results in a scalar 8-LSB (Value) 8-MSB (Index)
max - Calculate the max value and index between all the vector elements and a scalar put the value and the index is a scalar 8-LSB (Value) 8-MSB (Index)

Table 10: Vector to Scalar Instruction list

Symbol	Name	Operands
dot	Sum a vector and a scalar	Sd,Vm,Sn
min	Min and Min index of a vector	Sd,Vm,Sn Sd.l - Value Sd.h - Index Sn.l - Value Sn.h - Index
max	Max and Max index of a vector	Sd,Vm,Sn Sd.l - Value Sd.h - Index Sn.l - Value Sn.h - Index

6. Permutations

The following table summarize the vector rotation for each of the permutation.

Table 11: Permutation operation

	Name	Operand supported	Equation
vcs	Vector compare select		
shup	Shift Up	<Sm/Imm> Vd,Vm Sm Value,Imm = {1,2}	V[0] = 0 V[n] = V[n-1] n > 0
shup1w	Shift Up 1 and Wrap	Vd,Vm,Sm,Sn	V[0] = S0.S15 S0.S15 = V[31] V[n] = V[n-1] n > 0
shdn	Shift down	<Sm/Imm> Vd,Vm Sm Value,Imm = {1,2}	V[n] = V[n+1] n<31 V[31] = 0
shdn1w	Shift down 1 and wrap	Vd,Vm,Sm,Sn	V[n] = V[n+1] n<31 V[31] = S0.S15 S0.S15 = V[0]
hfup	Shift half up	<Sm/Imm> Vd,Vm Sm Value,Imm = {1,2,4,8,16}	
hfdn	Shift half down	<Sm/Imm> Vd,Vm Sm Value,Imm = {1,2,4,8,16}	
bfly	Butterfly	<Sm/Imm> Vd,Vm Sm Value,Imm = {1,2,4,8,16}	
shrk	Shrink	<Sm/Imm> Vd,Vm Sm Value,Imm = {1,2,4,8,16}	
exp	Expand	<Sm/Imm> Vd,Vm Sm Value,Imm = {1,2,4,8,16}	

7. AGU

The AGU instructions move data from memory into register file. Based on the AGU register indexing.

7.1 Data addressing

Each AGU register contains 3 register BASE,ADDRESS and CONTROL.
The Base and the control registers are map in the special purpose space and can be access with trfs instruction. While the Address is mapped in the AGU register space. In AGU instruction only the AGU register is specified and the its control and based register are implicitly used.

Effective address = (Address&Mask+Base)
Address = Address+offset

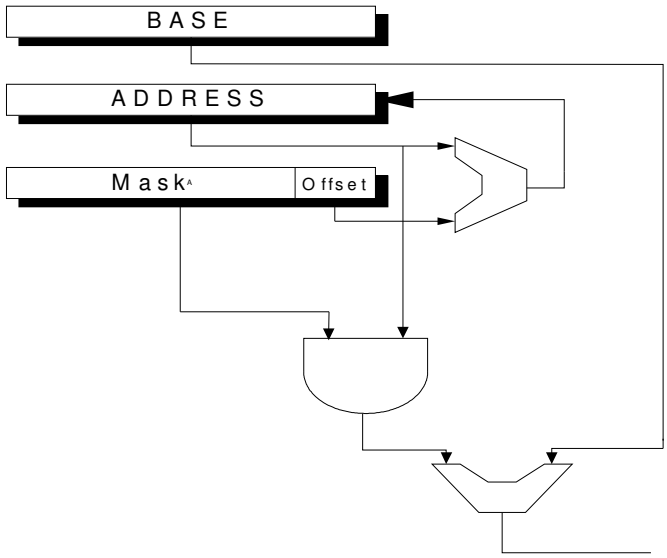


Table 12: AGU registers

	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Base			s/v[a]	Base Address												0
Address				Address												0
Control	Offset			Mask												Res

a. Scale/Vector

Table 13: AGU Control flags

Offset	000 - 0 001 - 2 010 - 4 011 - 8 100 - 16 101 - 32 110 - 64 111 - 128
Reserved	N/A

7.2 AGU instructions

Table 14: Transfer Instructions operands

Operands	Operands	Description
nill	-	No Operation
move.b	move.b Sd,(An.s)+/-	- Move byte from scalar memory to scalar register
	move.b Vd,(An.s)+/-	- Move byte from scalar memory expend it and move it to vector register
	move.b Ad,(An.s)+/-	- Move Byte from the memory into AGU register
	move.b (An.s)+/-,Sn	- Move register LSB register Sn to scalar memory
	move.b (An.s)+/-,Ad	- Move AGU 8-bit LSB to scalar memory

Table 14: Transfer Instructions operands

Operands	Operands	Description
move.w	move.w Sd,(An.s)+/-	Move 2B from scalar memory to scalar register
	move.w Ad,(An.s)+/-	Move 2B from the memory into AGU register
	move.w (An.s)+/-,Sn	- Move register Sn to scalar memory
	move.w (An.s)+/-,Am	- Move AGU to scalar memory
move.l	move.l Pd,(An.s)+/-	- Move 32bit from memory to predication register
	move.l (An.s)+/-,Sm:Sn	- Move 2 registers to scalar memory
move.2l	move.2l Pd:Pd+1,(An.s)+/-	- Move 64bits into two predicators
move.q	move.q Vn,(An.s)+/-	- Move 64bits from scalar memory into scalar vector queue
move.v	move.v Vd,(An.v)+/- move.v (An.v)+-,Vm	Move from vector memory to vector RF
	move.v Sd,(An.v)+/-,Sm	Move Vector from memory indexed by Scalar (Sm) into scalar
	move.v Ad,(An.v)+/-,Sm	Move Vector from memory indexed by scalar (Sm) into AGU
	move.v Pd,(An.v)+/-	Move Vector LSP into predictor

Examples

Modes:

3.3.1 16bit+16bit takes 2 cycles; r1,r0 Num1; r3,r2 Num2; r5,r4 Result

(1) add r4,r0,r2

(2) add.c r5,r1,r3

3.3.2 8x16 truncate to 16bits takes 2 cycles as follow:

// r1,r0 Num1; r2 Num2; r5,r4 Result

(1) mul r0,r2,r4

(2) mul.c r1,r2,r5

3.3.3 16x16 multiplication takes 5 cycles as follow:

// r1,r0 Num1; r3,r2 Num2; r7,r6,r5,r4 Result

(1) mul r0,r2,r4

(2) mul.c r1,r2,r5

(3) mull.c r3,r0,r5

(4) mull.c r3,r1,r6

(5) mul.c #0,r1,r7

Appendix A:
SPR memory map

Table 15: Special purpose register memory map

Address	Register	Description
0x0	L0_Start	Loop0 start register
0x2	L0_Size	Loop0 count register
0x4	L0_End	Loop0 end register
0x8	L1_Start	Loop1 start register
0x10	L1_Size	Loop1 count register
0x12	L1_End	Loop1 end register
0x20	AGU0_BASE	
0x22	AGU0_CNTR	
0x24	AGU1_BASE	
0x26	AGU1_CNTR	
0x28	AGU2_BASE	
0x2A	AGU2_CNTR	
0x2C	AGU3_BASE	
0x2E	AGU3_CNTR	
0x100-0x200	DMA	

** Extract instruction for scalar???

** Add Two guard bits on the vector allows, 4 additions before saturation
This might requires a special move that takes all 10bits to scalar.