/Sasha / 09-Sep-99

APEmille assembler instructions.

TARZAN Arithmetic operations

TADD reg3 reg1 reg2	[reg3] <= [reg1]+[reg2]	3(1)
TSUB reg3 reg1 reg2	[reg3] <= [reg1]-[reg2]	3
TMUL reg3 reg1 reg2	[reg3] <= [reg1]x[reg2] $_{\text{(31 LSB's)}}$	5
TDIV $reg3_1$ $reg3_2$ reg1 reg2	[$reg3_1$] <= [reg1]/[reg2] , [$reg3_2$] <= rest	37
TMULA reg3 reg1 reg2	[reg3(31)] <= sign ([reg1(31,22:0)]x[reg2(31:0)]) [reg3(30:23)] <= [reg1(30:23)] [reg3(22:0)] <= SHR_{31} ([reg1(31,22:0)]x[reg2(31:0)])	5
TADD3 reg3 reg0 reg1 reg2	[reg3] <= [reg0]+[reg1]+[reg2]	3
TSUB3 reg3 reg0 reg1 reg2	[reg3] <= [reg0]+[reg1]-[reg2]	3
TAGU1 reg4 disp	[reg4] <= disp	2
TAGU2 reg4 disp.reg0	[reg4] <= disp+[reg0]	3
TAGU3 reg4 disp.reg0.reg1	[reg4] <= disp+[reg0]+[reg1]	3

TARZAN logic and bitwise operations

TAND reg3 reg1 reg2	[reg3] <= [reg1]AND[reg2]	3
TOR reg3 reg1 reg2	[reg3] <= [reg1]OR[reg2]	3
TXOR reg3 reg1 reg2	[reg3] <= [reg1]XOR[reg2]	3
TNAND reg3 reg1 reg2	[reg3] <= [reg1]NAND[reg2]	3
TNOR reg3 reg1 reg2	[reg3] <= [reg1]NOR[reg2]	3
TXNOR reg3 reg1 reg2	[reg3] <= [reg1]XNOR[reg2]	3
TASH reg3 reg1 reg2	[reg3] <= ASH $_{[reg2]}$[reg1] (Restriction: -32<[reg2] <32)	3
TLSH reg3 reg1 reg2	[reg3] <= LSH $_{[reg2]}$[reg1] (Restriction: -32<[reg2] <32)	3
TROT reg3 reg1 reg2	[reg3] <= LSH $_{[reg2]}$[reg1]	3

Flow control (Tarzan) instructions

TEQ reg1 reg2	if([reg1]=[reg2]) then Flag <='1'	4
TNE reg1 reg2	if([reg1]!=[reg2]) then Flag <='1'	4
TGT reg1 reg2	if([reg1]>[reg2]) then Flag <='1'	4
TLT reg1 reg2	if([reg1]<[reg2]) then Flag <='1'	4
TGE reg1 reg2	if([reg1]>=[reg2]) then Flag <='1'	4
TLE reg1 reg2	if([reg1]=<[reg2]) then Flag <='1'	4
TINCCOMP reg0 reg1 reg2	if(([reg0]+[reg1])>[reg2]) then Flag <='1' [reg0] <= zero_disp + [reg0] +[reg1]	4

(1) length of a microcommand in number of PM words.

TANY	if(Gif='1') then Flag <='1'	1
TNONE	if(Gif='0') then Flag <='1'	1
TCNB	if(CNB='1') then Flag <='1'	1
TGETPC reg4	[reg4] <= PC	2
JUMP *label.reg0.(reg1)	[PC] <= *label+[reg0] (+[reg1])	1

JUMPIF *label.reg0.(reg1)	If(Flag ='1') then [PC] <= *label+[reg0] (+[reg1])
	else [PC] <= [PC]+1
		1

JUMPIFNOT *label.reg0.(reg1)	If(Flag ='0') then [PC] <= *label+[reg0] (+[reg1])
	else [PC] <= [PC]+1
		1

GOSUB *label.reg0.(reg1) reg4	[reg4] <= [PC] [PC] <= *label + [reg0] (+[reg1])	10
LABEL *label	Symbolic reference to an address of a following instruction	
BREAK	Soft Exception	
HALT halt_code(4-bits)		

JANE single precision arithmetics (floating point 32 bist).

JSNORM_$\alpha\beta$ reg3 reg0 reg1 reg2	[reg3] <= α ([reg0]x[reg1] + β[reg2]) Where α, β = P(+),M(-)	zn[2]	10	if[3]
JSNORM_Aβ reg3 reg0 reg1 reg2	[reg3] <= ABS([reg0]x[reg1] + β[reg2]) Where β = P(+),M(-)	z?	10	if
JSADD3_$\alpha\beta\gamma$ reg3 reg0 reg1 reg2	[reg3] <= α ([reg0] + β[reg1] + γ[reg2]) Where α, β, γ = P(+),M(-)	zn	10	if
JSADD3_A$\beta\gamma$ reg3 reg0 reg1 reg2	[reg3] <= ABS ([reg0] + β[reg1] + γ[reg2]) Where β, γ = P(+),M(-)	z?	10	if
JVNORM_$\alpha\beta$ reg3 reg0 reg1 reg2	[reg3] <= α ([reg0]x[reg1] + β[reg2]) [reg3+1] <= α ([reg0+1]x[reg1+1] + β[reg2+1]) where α, β = P(+),M(-) ; reg0,1,2,3 are even	zn	12	if
JVNORM_Aβ reg3 reg0 reg1 reg2	[reg3] <= ABS([reg0]x[reg1] + β[reg2]) [reg3+1] <= α ([reg0+1]x[reg1+1] + β[reg2+1]) where β = P(+),M(-) ; reg0,1,2,3 are even	zn	12	if

JANE complex single precision arithmetics(floating point 32 bist).

JCNORM_ $\alpha\beta$ reg3 reg0 reg1 reg2	$[reg3]_C$ <= α ($[reg0]_C$ x $[reg1]_C$ + $\beta[reg2]_C$) where α, β = P(+),M(-); reg0,1,2,3 are even; $[reg]_C$=[reg] +i[reg+1] - complex	z	12	if
JCNORM_ Aβ reg3 reg0 reg1 reg2	$[reg3]_C$ <= (ABS (REAL ($[reg0]_C$ x $[reg1]_C$ + $\beta[reg2]_C$) + i * ABS (IMAGE ($[reg0]_C$ x $[reg1]_C$ + $\beta[reg2]_C$)) where α, β = P(+),M(-); reg0,1,2,3 are even; $[reg]_C$=[reg] +i[reg+1] - complex	z	12	if

JANE double precision arithmetics(floating point 64 bits).

JDNORM_ $\alpha\beta$ reg3 reg0 reg1 reg2	$[reg3]_D$ <= α ($[reg0]_D$ x$[reg1]_D$ + $\beta[reg2]_D$) Where α, β = P(+),M(-) ;reg0,1,2,3 are even; $[reg]_D$ = [reg+1] &[reg]	zn	12	if
JDNORM_A $\alpha\beta$ reg3 reg0 reg1 reg2	$[reg3]_D$ <= ABS ($[reg0]_D$ x$[reg1]_D$ + $\beta[reg2]_D$) Where α, β = P(+),M(-) ;reg0,1,2,3 are even; $[reg]_D$ = [reg+1] &[reg]	z?	12	if

JANE integer arithmetics (interger 32 bits).

JIADD reg3 reg0 reg1	[reg3] <= [reg0] + [reg1]	zn	4	if
JISUB reg3 reg0 reg1	[reg3] <= [reg0] - [reg1]	zn	4	if
JIMULT reg3 reg0 reg1	[reg3] <= [reg0] x [reg1]	zn	10	if
JILEADONE reg3 reg0	[reg3] <= INTEGER ($\log_2$ [reg0])	zn	4	if
JIBITCOUNT reg3 reg0	[reg3] <= number of 1's in [reg0]		4	if

JANE logic and bitwise operations

JIAND reg3 reg0 reg1	[reg3] <= [reg0] AND [reg1]	zn	4	if
JINAND reg3 reg0 reg1	[reg3] <= [reg0] NAND [reg1]	zn	4	if
JIOR reg3 reg0 reg1	[reg3] <= [reg0] OR [reg1]	zn	4	if

(2) condition outputs of the FILU affected by the command.
(3) writing the result to the destination may be conditioned.

JIXOR reg3 reg0 reg1	[reg3] <= [reg0] XOR [reg1]	zn	4	if
JILSH reg3 reg0 reg1	[reg3] <= LSH$_{[reg1]}$ [reg0]	zn	4	if
JIASH reg3 reg0 reg1	[reg3] <= ASH$_{[reg1]}$ [reg0]	zn	4	if
JIROT reg3 reg0 reg1	[reg3] <= ROT$_{[reg1]}$ [reg0]	zn	4	if

JANE LUT operations (fetching a value or a seed of the requested value).

JSLUTONE reg4	[reg4] <= single precision ONE		1	if
JSLUTLSB reg4$_i$ reg4$_o$	[reg4$_i$]$_S$ <= single precision LSB [reg4$_o$]		1	if
JSLUTINV reg4$_i$ reg4$_o$	[reg4$_i$] <= seed (1/[reg4$_o$])		3	if
JSLUTINVSQRT reg4$_i$ reg4$_o$	[reg4$_i$] <= seed (1/ [reg4$_o$]$^{1/2}$)		3	if
JSLUTMOVE reg4$_i$ reg4$_o$	[reg4$_i$] <= [reg4$_o$]		3	if
JSLUTCHS reg4$_i$ reg4$_o$	[reg4$_i$] <= - [reg4$_o$]		3	if
JSLUTEXP reg4$_i$ reg4$_o$	[reg4$_i$] <= seed (e$^{[\text{ reg4o}]}$)		3	if
JSLUTLOGM reg4$_i$ reg4$_o$	[reg4$_i$] <= 1st_seed (log$_e$ [reg4$_o$])		3	if
JSLUTLOGE reg4$_i$ reg4$_o$	[reg4$_i$] <= 2nd_seed (log$_e$ [reg4$_o$])		3	if
JDLUTONE reg4	[reg4]$_D$ <= double precision ONE		1	if
JDLUTLSB reg4$_i$ reg4$_o$	[reg4$_i$]$_D$ <= double precision LSB [reg4$_o$]		1	if
JDLUTINV reg4	[reg4]$_D$ <= seed (1/[reg4]$_D$)		3	if
JDLUTINVSQRT reg4	[reg4]$_D$ <= seed (1/ [reg4]$_D{}^{1/2}$)		3	if
JDLUTMOVE reg4$_i$ reg4$_o$	[reg4$_i$]$_D$ <= [reg4$_o$]$_D$		3	if
JDLUTCHS reg4$_i$ reg4$_o$	[reg4$_i$]$_D$ <= - [reg4$_o$]$_D$		3	if
JDLUTEXP reg4$_i$ reg4$_o$	[reg4$_i$]$_D$ <= seed (e$^{[\text{ reg4o}]}{}_D$)		3	if
JDLUTLOGM reg4$_i$ reg4$_o$	[reg4$_i$]$_D$ <= 1st_seed (log$_e$ [reg4$_o$]$_D$)		3	if
JDLUTLOGE reg4$_i$ reg4$_o$	[reg4$_i$]$_D$ <= 2nd_seed (log$_e$ [reg4$_o$]$_D$)		3	if
JVLUTONE reg4	[reg4] <= single precision ONE [reg4+1] <= single precision ONE		1	if
JVLUTLSB reg4$_i$ reg4$_o$	[reg4$_i$] <= single precision LSB [reg4$_o$] [reg4$_i$ +1] <= single precision LSB [reg4$_o$ +1]		1	if
JVLUTINV reg4$_i$ reg4$_o$	[reg4$_i$] <= seed (1/[reg4$_o$]) [reg4$_i$ +1] <= seed (1/[reg4$_o$ +1])		3	if
JVLUTINVSQRT reg4$_i$ reg4$_o$	[reg4$_i$] <= seed (1/ [reg4$_o$] $^{1/2}$) [reg4$_i$ +1] <= seed (1/ [reg4$_o$ +1] $^{1/2}$)		3	if
JVLUTMOVE reg4$_i$ reg4$_o$	[reg4$_i$] <= [reg4$_o$] [reg4$_i$ +1] <= [reg4$_o$+1]		3	if
JVLUTCHS reg4$_i$ reg4$_o$	[reg4$_i$] <= - [reg4$_o$] [reg4$_i$ +1] <= - [reg4$_o$+1]		3	if
JVLUTEXP reg4$_i$ reg4$_o$	[reg4$_i$] <= seed (e$^{[\text{ reg4o}]}$) [reg4$_i$ +1] <= seed (e$^{[\text{ reg4o+1}]}$)		3	if
JVLUTLOGM reg4$_i$ reg4$_o$	[reg4$_i$] <= 1st_seed (log$_e$ [reg4$_o$]) [reg4$_i$ +1] <= 1st_seed (log$_e$ [reg4$_o$ +1])		3	if

JVLUTLOGE reg4_i reg4_o	[reg4_i] <= 2^{nd}_seed (log $_e$ [reg4_o]) [reg4_i +1] <= 2^{nd}_seed (log $_e$ [reg4_o +1])		3	if

JANE type conversion instructions

JSTOD reg3 reg0	[reg3]$_D$ <= convert ([reg0]$_S$)	zn	12	if
JDTOS reg3 reg0	[reg3]$_S$ <= convert ([reg0]$_D$)	zn	12	if
JSTOI reg3 reg2	[reg3]$_i$ <= convert ([reg2]$_S$)	zn	10	if
JDTOI reg3 reg2	[reg3]$_i$ <= convert ([reg2]$_D$)	zn	10	if
JITOS reg3 reg2	[reg3]$_S$ <= convert ([reg2]$_i$)	zn	10	if
JITOD reg3 reg2	[reg3]$_D$ <= convert ([reg2]$_i$)	zn	10	if

JANE local boolean conditions and IF stack

JSPUSHEQ reg0 reg1	PushStack ; TopStack <= ([reg0]$_S$ = [reg1]$_S$);	11
JSPUSHNE reg0 reg1	PushStack ; TopStack <= ([reg0]$_S$!= [reg1]$_S$);	11
JSPUSHGT reg0 reg1	PushStack ; TopStack <= ([reg0]$_S$ > [reg1]$_S$);	11
JSPUSHLT reg0 reg1	PushStack ; TopStack <= ([reg0]$_S$ < [reg1]$_S$);	11
JSPUSHGE reg0 reg1	PushStack ; TopStack <= ([reg0]$_S$ >= [reg1]$_S$);	11
JSPUSHLE reg0 reg1	PushStack ; TopStack <= ([reg0]$_S$ =< [reg1]$_S$);	11
JCPUSHEQ reg0 reg1	PushStack ; TopStack <= ([reg0]$_C$ = [reg1]$_C$); Reg0,1 are even.	13
JCPUSHNE reg0 reg1	PushStack ; TopStack <= ([reg0]$_C$!= [reg1]$_C$); Reg0,1 are even.	13
JDPUSHEQ reg0 reg1	PushStack ; TopStack <= ([reg0]$_D$ = [reg1]$_D$); Reg0,1 are even	13
JDPUSHNE reg0 reg1	PushStack ; TopStack <= ([reg0]$_D$!= [reg1]$_D$); Reg0,1 are even	13
JDPUSHGT reg0 reg1	PushStack ; TopStack <= ([reg0]$_D$ > [reg1]$_D$); Reg0,1 are even	13
JDPUSHLT reg0 reg1	PushStack ; TopStack <= ([reg0]$_D$ < [reg1]$_D$); Reg0,1 are even	13
JDPUSHGE reg0 reg1	PushStack ; TopStack <= ([reg0]$_D$ >= [reg1]$_D$); Reg0,1 are even	13
JDPUSHLE reg0 reg1	PushStack ; TopStack <= ([reg0]$_D$ =< [reg1]$_D$); Reg0,1 are even	13
JIPUSHEQ reg0 reg1	PushStack ; TopStack <= ([reg0] $_i$ = [reg1] $_i$);	5
JIPUSHNE reg0 reg1	PushStack ; TopStack <= ([reg0] $_i$!= [reg1] $_i$);	5
JIPUSHGT reg0 reg1	PushStack ; TopStack <= ([reg0] $_i$ > [reg1] $_i$);	5
JIPUSHLT reg0 reg1	PushStack ; TopStack <= ([reg0] $_i$ < [reg1] $_i$);	5
JIPUSHGE reg0 reg1	PushStack ; TopStack <= ([reg0] $_i$ >= [reg1] $_i$);	5
JIPUSHLE reg0 reg1	PushStack ; TopStack <= ([reg0] $_i$ =< [reg1] $_i$);	5
JSTKREPUSH	PushStack ; TopStack <= TopStack	1
JPOP	PopStack	1
JSTKAND	TopStack <= TopStack AND uTopStack; PopStack	1

	Where uTopStack - a stack element under the TopStack.	
JSTKOR	TopStack <= TopStack OR uTopStack; PopStack	1
JSTKXOR	TopStack <= TopStack XOR uTopStack; PopStack;	1
JSTKRESET	ClearStack	1
JSTKANDBIS	TopStack <= TopStack AND uTopStack	1

JSTKSWAP	If StackEmpty TopStack <= ‘0’ , tempTop <= ‘0’, PushStack;
	elsif StackUno TopStack<=tempTop , tempTop <=TopStack ;
	else TopStack<=uTopStack, uTopStack<=TopStack .
		1

JSTKNOT	TopStack <= NOT TopStack	1

Data transfer instructions

MEMTOT1 treg4 disp	[treg4] <= TzDM[disp]	7 if
MEMTOT2 treg4 disp.treg0	[treg4] <= TzDM[disp+[treg0]]	7 if
MEMTOT3 treg4 disp.treg0.treg1	[treg4] <= TzDM[disp+[treg0] +[treg1]]	7 if
TTOMEM1 disp treg4	TzDM[disp] <= [treg4]	2 if
TTOMEM2 disp.treg0 treg4	TzDM[disp+[treg0]] <= [treg4]	2 if
TTOMEM3 disp.treg0.treg1 treg4	TzDM[disp+[treg0] +[treg1]] <= [treg4]	2 if
MEMTOJ1 jreg4: N disp	[jreg4 + i] <= JnDM[disp + LOF + i], i=0,…(N-1) Where (disp + LOF) and jreg4 must be both even or odd.	12+N/2 if
MEMTOJ2 jreg4: N disp.treg0	[jreg4 + i] <= JnDM[disp + [treg0] + LOF + i], i=0,…(N-1) Where (disp+[treg0]+LOF) and jreg4 must be both even or odd.	12+N/2 if
MEMTOJ4 jreg4: N disp.treg0.treg1	[jreg4 + i] <= JnDM[disp + [treg0] + [treg1] + LOF + i], i=0,…(N-1) Where (disp+[treg0]]+[treg1]+LOF) and jreg4 must be both even or odd.	12+N/2 if
JTOMEM1 disp jreg4: N	JnDM[disp + LOF + i] <= [jreg4 + i], i=0,…(N-1) Where (disp + LOF) and jreg4 must be both even or odd.	8+N/2 if
JTOMEM2 disp.treg0 jreg4: N	JnDM[disp + [treg0] + LOF + i] <= [jreg4 + i], i=0,…(N-1) Where (disp+[treg0]+LOF) and jreg4 must be both even or odd.	8+N/2 if
JTOMEM3 disp.treg0.treg1 jreg4: N	JnDM[disp + [treg0] + [treg1] + LOF + i] <= [jreg4 + i], i=0,…(N-1) Where (disp+[treg0]+[treg1]+LOF) and jreg4 must be both even or odd.	8+N/2 if
TTOJ jreg4 disp.treg0.treg1	[jreg4] <= disp + [treg0] + [treg1]	5 if
JTOT treg4 jreg4	translated to consecutive JTOC jreg4 and CTOT treg4	14 if
JTOC jreg4	ch_xreg(i) <= [jreg4] Where i= 0..7 corresponds to the Jane numbers.	6 if
JTOC0 jreg4	ch_xreg(i) <= 1st B[jreg4], 1st B[jreg4], 1st B[jreg4], 1st B[jreg4] Where i= 0..7 corresponds to the Jane numbers.	6 if
JTOC1 jreg4	ch_yreg(i) <= 2nd B[jreg4], 2nd B[jreg4], 2nd B[jreg4], 2nd B[jreg4] Where i= 0..7 corresponds to the Jane numbers.	6 if
JTOC2 jreg4	ch_zreg(i) <= 3d B[jreg4], 3d B[jreg4], 3d B[jreg4], 3d B[jreg4] Where i= 0..7 corresponds to the Jane numbers.	6 if
CTOT treg4	[treg4] <= ch_xreg(0)	7 if
JLOFRESET	LOF <= 0x00000000	1 if
JLOFSET jreg4	LOF <= [jreg4]	1 if

Memory initialisation.

TCONST tadr const	TzDM[tadr] <= const Restrictions: tadr =< 0x1FFFF.
JCONST jadr const	JnDM[jadr] <= const Restrictions: jadr =< 0x3FFF.
JCONSTD jadr {flp_const}	JnDM[jadr] <= flp_const Restrictions: jadr < 0x3FFF jadr - even.
JCONSTV jadr const1 const2	JnDM[jadr] <= flp_const1 JnDM[jadr+1] <= flp_const2 Restrictions: jadr < 0x3FFF jadr - even.