ADChips® AE32000 Instruction Set
Quick Reference Card

Key to Tables	
%Rc	Shift count register (GPR)
%Ri	Index register (GPR)
%Rs	Source register (GPR)
%Rd	Destination register (GPR)
imm	Immediate value (Maximum 32bit)
imm#	Immediate value (Maximum #bit)

<label>	Assembler label
/	Exclusively used
|	Concatenation
@unit	at unit
C	Carry flag
[address]	32bit data from address

Operation		V	Assembler	SR Mod.	Action	T	L	LD	E
Move	Move		MOV %Rs, %Rd		%Rd = %Rs	△	△	O	O
	with add		LEA (%Rs/%SP, imm), %Rd/%SP		%Rd/%SP = %Rs/%SP + imm				
	from GPR to MH		MTMH %Rs		%MH = %Rs				
	from GPR to ML		MTML %Rs		%ML = %Rs				
	from MH to GPR		MFMH %Rd		%Rd = %MH				
	from ML to GPR		MFML %Rd		%Rd = %ML				
	from GPR to MRE		MTMRE %Rs		%MRE = %Rs				
	from MRE to GPR		MFMRE %Rd		%Rd = %MRE				
	from GPR to CR0		MTCR0 %Rs		%CR0 = %Rs				
	from GPR to CR1		MTCR1 %Rs		%CR1 = %Rs				
	from CR0 to GPR		MFCR0 %Rd		%Rd = %CR0				
	from CR1 to GPR		MFCR1 %Rd		%Rd = %CR1				
	imm to GPR		LDI imm, %Rd		%Rd = imm				
Arithmetic	Add		ADD %Rs/imm, %Rd	C Z S V	%Rd = %Rd + %Rs/imm	O	O	O	O
	with short immediate		ADDQ imm5, %Rd	C Z S V	%Rd = %Rd + imm_5				
	with carry		ADC %Rs/imm, %Rd	C Z S V	%Rd = %Rd + %Rs/imm + C				
	Subtract		SUB %Rs/imm, %Rd	C Z S V	%Rd = %Rd - %Rs/imm				
	with carry		SBC %Rs/imm, %Rd	C Z S V	%Rd = %Rd - %Rs/imm – C				
Logical	Test		TST %Rs1/imm, %Rs2	Z S	Update SR flag on %Rs1 AND %Rs2	O	O	O	O
	AND		AND %Rs/imm, %Rd	Z S	%Rd = %Rd AND %Rs/imm				
	OR		OR %Rs/imm, %Rd	Z S	%Rd = %Rd OR %Rs/imm				
	XOR		XOR %Rs/imm, %Rd	Z S	%Rd = %Rd XOR %Rs/imm				
	NOT		NOT %Rd	Z S	%Rd = bit inverse(%Rd)				
Compare	Compare		CMP %Rs1/imm, %Rs2	C Z S V	Update SR flag on %Rs2 - %Rs1	O	O	O	O
	with short immediate		CMPQ imm5, %Rs	C Z S V	Update SR flag on %Rs – imm5				
Shift	Arithmetic shift right		ASR %Rc/imm_5, %Rd	C Z S	{(%Rc/imm_5}(Sign(%Rd)), %Rd >> (%Rc/imm_5)}	O	O	O	O
	Logical shift right		LSR %Rc/imm_5, %Rd	C Z S	{(%Rc/imm_5}(0) , %Rd >> (%Rc/imm_5)}				
	Arithmetic shift left		ASL %Rc/imm_5, %Rd	C Z S	{%Rd << (%Rc/imm_5), (%Rc/imm_5)(0)}				
	Set shift left		SSL %Rc/imm_5, %Rd	C Z S	{%Rd << (%Rc/imm_5), (%Rc/imm_5)(1)}				
Multiply	Multiply		MUL %Rs/imm, %Rd		%MH|%ML = %Rs * %Rd, %Rd = %ML	O	O	O	O
	accumulate		MAC %Rs1/imm, %Rs2		%MH|%ML = (%Rs1 * %Rs2) + %MH|%ML				
	unsigned		MULU %Rs/imm, %Rd		%MH|%ML = unsigned(%Rs * %Rd), %Rd = %ML				

Misc	Extension from byte to word		`EXTB %Rd`	Z S	SignExtent(%Rd[7:0])	O	O	O	O
	Extension from short to word		`EXTS %Rd`	Z S	SignExtent(%Rd[15:0])				
	Convert from word to byte		`CVB %Rd`	Z S	%Rd = %Rd AND 0xff				
	Convert from word to short		`CVS %Rd`	Z S	%Rd = %Rd AND 0xffff				
	Count leading zero		`CNT0 %Rs`	Z	%R0 = number of leading zeroes in %Rs				
	Count leading one		`CNT1 %Rs`	Z	%R0 = number of leading ones in %Rs				
DSP Acceleration	MAC Word		`MAC %Rs1/imm, %Rs2`		Reference AE32000-isa-rm_ko.pdf	X	X	O	O
	Short SIMD		`MACS %Rs1/imm, %Rs2`						
	Byte SIMD		`MACB %Rs1/imm, %Rs2`						
	Multiple Sum of Product (Short)		`MSOPS %Rs1/imm, %Rs2`						
	Multiple Sum of Product (Byte)		`MSOPB %Rs1/imm, %Rs2`						
	Saturate Add Word		`SADD %Rs/imm, %Rd`						
	Short (signed)		`SADDS %Rs/imm, %Rd`						
	Short (unsigned)		`SADUS %Rs/imm, %Rd`						
	Byte (signed)		`SADDB %Rs/imm, %Rd`						
	Byte (unsigned)		`SADUB %Rs/imm, %Rd`						
	Unpack Short to high		`UNKHS %Rsrc_grp, %Rd`						
	Short to low		`UNKLS %Rsrc_grp, %Rd`						
	Byte from 0 to high		`UNK0HS %Rsrc_grp, %Rd`						
	Byte from 0 to low		`UNK0LS %Rsrc_grp, %Rd`						
	Byte from 1 to high		`UNK1HS %Rsrc_grp, %Rd`						
	Byte from 1 to low		`UNK1LS %Rsrc_grp, %Rd`						
	Byte from 2 to high		`UNK2HS %Rsrc_grp, %Rd`						
	Byte from 2 to low		`UNK2LS %Rsrc_grp, %Rd`						
	Byte from 3 to high		`UNK3HS %Rsrc_grp, %Rd`						
	Byte from 3 to low		`UNK3LS %Rsrc_grp, %Rd`						
	Average SIMD Short		`AVGS %Rs/imm, %Rd`						
	SIMD Byte		`AVGB %Rs/imm, %Rd`						
	Rotate Left		`ROL imm, %Rd`						
	Right		`ROR imm, %Rd`						
	Mininum		`MIN %Rs/imm, %Rd`						
	Maximum		`MAX %Rs/imm, %Rd`						
	Absolute Value		`ABS %Rd`						
	Fixed Point Multiply Result Shift		`MRS imm, %Rd`						
Branch	Jump		`JMP <label>`		PC = address of <Label>	O	O	O	O
	and link		`JAL <label>`		LR = PC, PC = address of <Label>				
	on overflow clear		`JNV <label>`		if (V == 0) PC = address of <Label>				
	on overflow set		`JV <label>`		if (V == 1) PC = address of <Label>				
	on sign clear / positive or zero		`JP <label>`		if (S == 0) PC = address of <Label>				
	on sign set / negative		`JM <label>`		if (S == 1) PC = address of <Label>				
	on non-zero / not equal		`JNZ <label>`		if (Z == 0) PC = address of <Label>				
	on zero / equal		`JZ <label>`		if (Z == 1) PC = address of <Label>				
	on carry clear / unsigned higher or equal		`JNC <label>`		if (C == 0) PC = address of <Label>				
	on carry set / unsigned lower		`JC <label>`		if (C == 1) PC = address of <Label>				
	on signed greater		`JGT <label>`		if ((Z+S^V) == 0) PC = address of <Label>				
	on signed less		`JLT <label>`		if ((S^V) == 1) PC = address of <Label>				
	on signed greater or equal		`JGE <label>`		if ((S^V) == 0) PC = address of <Label>				

	on signed less or equal		JLE <label>		if ((Z+S^V) == 1) PC = address of <Label>				
	on unsigned higher		JHI <label>		if ((C+Z) == 0) PC = address of <Label>				
	on unsigned lower or equal		JLS <label>		if ((C+Z) == 1) PC = address of <Label>				
	register indirect		JR %Rs		PC = %Rs				
	register indirect and link		JALR %Rs		LR = PC, PC = %Rs				
	to link register		JPLR		PC = LR				
Load	Word		LD (%Ri/%SP, imm), %Rd		%Rd = [%Ri/%SP + imm]	△	△	O	O
	Byte		LDBU (%Ri/%SP, imm), %Rd		%Rd = ZeroExtent[Byte from (%Ri/%SP + imm)]				
	signed		LDB (%Ri/%SP, imm), %Rd		%Rd = SignExtent[Byte from (%Ri/%SP + imm)]				
	Half word (short)		LDSU (%Ri/%SP, imm), %Rd		%Rd = ZeroExtent[Short from (%Ri/%SP + imm)]				
	signed		LDS (%Ri/%SP, imm), %Rd		%Rd = SignExtent[Short from (%Ri/%SP + imm)]				
	Auto Incremental		LDAU CRNO, %Ri, %Rd		%Rd = [%Ri + offset@CRNO]				
Load Multiple	Pop		POP <reg list>		while (all regs in reg list is poped) <lower num unpoped register in reg list> = [%SP], SP = SP + 4				
Store	Word		ST %Rs, (%Ri/%SP, imm)		[%Ri/%SP + imm] = %Rs	△	△	O	O
	Byte		STB %Rs, (%Ri/%SP, imm)		[%Ri/%SP + imm][7:0] = %Rs[7:0]				
	Half word (short)		STS %Rs, (%Ri/%SP, imm)		[%Ri/%SP + imm][15:0] = %Rs[15:0]				
	Auto Incremental		STAU CRNO, %Ri, %Rs		[%Ri + offset@CRNO] = %Rs				
Store Multiple	Push		PUSH <reg list>		while (all regs in reg list is poped) [%SP = %SP -4] = <higher num unpushed register in reg list>				
Coprocessor	Instruction		CPCMD coprocessor command		Send instruction(imm) to coprocessor	O	O	O	O
	Move to GPR from coproc		MVFC %Rs@CP		%R0 = %Rs@CP				
	Move to coproc from GPR		MVTC %Rd@CP		%Rd@CP = %R0				
	Load		LDC (%R0/%SP, imm), %Rd@CP		%Rd@CP = [%R0/%SP + imm]				
	Store		STC %Rs@CP, (%R0/%SP, imm)		[%R0/%SP + imm] = %Rs@CP				
Polling	Check status bit in copoc		GETC imm_4	Z	SR.zero_flag = %SR.bit<imm_4>@CP				
	Exception on coprocessor status		EXEC imm_4	Z	if (SR.zero_flag = %SR.bit<imm_4>@CP) CPEXEC occur				
NOP	No Operation		NOP		Bubble Cycle instruction	O	O	O	O
Soft Interrupt	Software Interrupt		SWI imm_4		Software interrupt processor exception	O	O	O	O
STEP			STEP		Sing step debugging	X	O	O	X
Halt			HALT imm_4		Halt for low power	O	O	O	O
Breakpoint			BRKPT		Prefetch abort and enter debug state	X	O	O	X
SR control	Set a bit in SR		SET imm_4		SR.bit<imm_4> = 1 depend on processor mode	O	O	O	O
	Clear a bit in SR		CLR imm_4		SR.bit<imm_4> = 0 depend on processor mode				
Synchronize			SYNC		Synchronize for critical section handle	O	O	O	O

* All immediate values use signed number except Shift Operations, SET, CLR, HALT, SWI and GETC/EXEC.

* T : AE32000C-Tiny

* L : AE32000C-Lucida/AE32000C-Lucifer

* LD : AE32000C-Lucida/AE32000C-Lucifer with DSP

* E : AE32000C-Empress