TMS320C5x to TMS320C54x Translation Utility

Literature Number: BPRA075
Texas Instruments Europe
February 1998

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Contents

List of Tables

TMS320C5x to TMS320C54x Translation Utility

1. Introduction

This document describes the utility 'LEADT.EXE' (or ‘LEADT’ for UNIX) for translating 'C5x assembly source files to 'C54x. The translator is not designed to be an automatic translation from one processor to another. There are differences in the architectures that can not be handled by the ‘intellect’ of this software utility but it does give the necessary warnings and errors to prompt the user in the correct direction.

A brief summary of the translator capability is as follows :

- LEADT only works at assembly (mnemonic) level and will translate instructions, operands, and directives.
- Will pass through comments, labels, and other untranslated assembly time structures.
- Assumes that the 'C5x input code passes through the 'C5x assembly without error. There is no (little) 'C5x syntax checking done.
- Output of LEADT should pass through the 'C54x assembler without error but will not necessarily be functional.
- Errors, warnings, and hints are output as a result of the translation process and can be found embedded in the translated output as well as the information file.
- An information file, LEADT.INF, is produced to give auxiliary information about the translation process.
- LEADT will translate from other fixed point family members such as 'C1x, 'C2x, and 'C2xx. However, these device translations have not been tested as has been the 'C5x process. Default supported translation is for the 'C5x only.
- LEADT is available for PC’s (DOS) and Sun’s (SUN-OS).

2. Translation Methods

There are a number of different approaches that can be adopted when translating from one architecture to another. The main types that can be deployed are :

- **Direct Instruction Mapping** : This is the method used by the LEADT utility. Original code sequences are analysed at the instruction level and categorised into instruction types. The target architecture is also mapped into the same categories and the closest functional match is found. This is not the optimum form of translation for the following reasons :
  - Unsupported instructions on the original architecture either map to a number of target instructions or will not be translated at all. Often, more instructions are required on the target to emulate the original processor.
  - Architectural enhancements on the target cannot be fully utilised. Increased efficiency on the target can then be achieved only by a faster cycle time or increased memory resource which are almost unrelated to the translation process.

- **High-Level Language :** This method involves the up-conversion to a high-level or algebraic language which is normally common to both architectures. If we take the ‘C’ language as an example, then the translation would involve the generation of ‘C’ source that describes that original assembler source in a bit-exact nature. Target code would then be generated through the target’s ‘C’ compiler. An indirect benefit of this approach is the portability of the code for any future processor; however the disadvantages are :
  - Reliance on the efficiency of the target ‘C’ compiler in terms of code size and cycle optimisation. Fixed point processors are generally not known for having efficient compilers ('C6x being exception to the rule).
  - Inefficiencies of the ‘C’ language in being able to describe fixed-point arithmetic and other processor specific bit-exact operations and structures.
- **Operation analysis :** This method is the most efficient method but also the most complex to implement. In its purest form it requires an operational analysis of the original code such that a primitive description can be produced. Mapping is then performed by grouping the primitives into the target instructions. This enables the usage of the target architecture’s enhancements leading to decreased instruction counts as well as cycles. This process is most easily performed when the target instruction set is closely matched to the primitive set, i.e. such as the 'C6x. The process of translating from either the 'C5x or 'C54x processors to the 'C6x could be done quite efficiently by the generation of linear assembly. The assembly optimiser would then be used for register and resource allocation, software pipelining and general architecture optimisations. It would be expected that the translator in this instance would offer a more automatic flow than LEADT.

3. Architecture Mapping

This section describes the architecture mapping used when performing the translation between the 'C5x and 'C54x. The mapping tables show the original 'C5x object with the 'C54x equivalent. The LEADT treatment column details how the translator processes the objects. Caveats are marked with superscript and commented at the bottom of the table. «Direct Translation» generally means that LEADT will map as per the original and target objects listed in the table.

Another table is included within section detailing the 'C54x architectural features that LEADT cannot or does not use. This is useful information in the translation process for the user when understanding the 'C54x resources available for optimising the ode further.

3.1 CPU register Mapping

Table 1: CPU Register Mapping

'C5x	'C54x	'C5x Field Description	LEADT Treatment
ACC	A	Accumulator	Direct Translation(2)
ACCB	B	Accumulator Buffer	Direct Translation(2)
AR0	AR0	Auxiliary Register 0	Direct Translation
AR1-AR7	AR1-AR7	Auxiliary Register 1-7	Direct Translation
ARCR	-	Auxiliary Register Compare Reg.	Invalid Register
BMAR	-	Block-move Address Reg.	Invalid Register
BRCR	BRC	Block Repeat Counter Reg.	Direct Translation
CBER1	-	Circular Buffer End Address 1	Invalid Register
CBER2	-	Circular Buffer End Address 2	Invalid Register
CBSR1	-	Circular Buffer Start Address 1	Invalid Register
CBSR2	-	Circular Buffer Start Address 2	Invalid Register
DBMR	-	Dynamic Bit Manipulation Reg.	Invalid Register
GREG	-	Global Memory Allocation Reg.	Invalid Register
IFR	IFR	Interrupt Flag Register	Direct Translation(1)
IMR	IMR	Interrupt Mask Register	Direct Translation(1)
INDX	-	Indirect Addressing index Reg.	Invalid Register
PAER	REA	Block Repeat End Address	Direct Translation
PASR	RSA	Block Repeat Start Address	Direct Translation
PMST	PMST	Processor-Mode-status Reg.	Direct Translation(1)
PREG	-	Product Register	Invalid Register(2)
RPTC	-	Repeat-counter Reg.	Invalid Register(2)
ST0	ST0	Status Register 0	Direct Translation(1) (2)
ST1	ST1	Status Register 1	Direct Translation(1) (2)
TREG0	T	Temporary Register 0	Direct Translation
TREG1	-	Temporary Register 1	Invalid Register
TREG2	-	Temporary Register 2	Invalid Register

(1) Bit positions and/or functionality change between 'C5x and 'C54x
(2) Register not referenced directly in assembler but is implicit for specific instruction

Table 2: Unused 'C54x CPU Registers

'C54x	'C54x Register Description
AG	A Register Guard Band
BG	B Register Guard Band
BK	Block Size Register
SP	Stack Pointer
TRN	Transition Register
XPC	Extended Addressing Reg. (548/549 only)

3.2 Peripheral Register Mapping

Peripheral registers differ according to the original and target processor family members. No single processor will contain all registers in the following table.

Table 3: Peripheral Register Mapping

'C5x	'C54x	'C5x Field Description	LEADT Treatment
ARR	ARR	Address Receive Register(BSP)	Direct Translation
AXR	AXR	Address Transmit Register(BSP)	Direct Translation
BKR	BKR	Receive Buffer-size Register(BSP)	Direct Translation
BKX	BKX	Transmit Buffer-size Register(BSP)	Direct Translation
BDXR	BDXR	Transmit Register(BSP)	Direct Translation
BDRR	BDRR	Receive Register(BSP)	Direct Translation
BSPC	BSPC	Serial Port Control (BSP)	Direct Translation
CWSR	-	WaitState Control Register	Invalid Register
DRR	DRR	Transmit Register(SP)	Direct Translation
DXR	DXR	Receive Register(SP)	Direct Translation
IOWSR	IOWSR	IO Waitstate Register	Direct Translation(1)
HPIC	HPIC	Host Port Control Register	Direct Translation
PA[0-15]	-	Memory mapped ports	Invalid Register(2)
PA[0-15]	[0x50-0x5F]	IN or OUT operand	Direct Translation(2)
PDWSR	SWWSR	Software Wait-state control Reg.	Direct Translation(1)
PRD	PRD	Timer Period Register	Direct Translation
SPC	SPC	Serial Port Control (SP)	Direct Translation
SPCE	SPCE	Serial Port Control Extension(BSP)	Direct Translation
TCSR	TCSR	Channel Select Register (TDM-SP)	Direct Translation
TCR	TCR	Timer Control Register	Direct Translation
TDXR	TDXR	Transmit Data Register (TDM-SP)	Direct Translation
TIM	TIM	Timer Counter Register	Direct Translation
TRAD	TRAD	Received Address Register (TDM-SP)	Direct Translation
TRCV	TRCV	Receive Data Register (TDM-SP)	Direct Translation
TRTA	TRTA	RX/TX Address Register (TDM-SP)	Direct Translation
TSPC	TSPC	Serial Port Control (TDM-SP)	Direct Translation

(1) Bit positions and/or functionality change between 'C5x and 'C54x
(2) Not translated if used with memory mapped access, only with IN or OUT instructions

Table 4: Unused 'C54x Peripheral Registers

'C54x	'C54x Register Description
ARR[0,1]	Address Receive Register(BSP)
AXR[0,1]	Address Transmit Register(BSP)
BSCR	Bank Switch Control Register
BDXR[0,1]	Transmit Register(BSP)
BDRR[0,1]	Receive Register(BSP)
BKR[0,1]	Receive Buffer-size Register(BSP)
BKX[0,1]	Transmit Buffer-size Register(BSP)
BSPC[0,1]	Serial Port Control (BSP)
SPCE[0,1]	Serial Port Control Extension(BSP)
CLKMOD	Clockmode Register

3.3 CPU Bit Field Mapping

The assembler and instruction sets allow for some bit fields to be accessed and modified explicitly as well as implicitly. The following table shows how the translation of these bit fields are handled by LEADT.

Table 5: CPU Bit-field Mapping

'C5x	'C54x	'C5x Field Description	LEADT Treatment
ARB	-	Auxiliary Register Pointer Buffer	Invalid Field(2)
ARP	ARP	Auxiliary Register Pointer	(2) (3)
AVIS	AVIS	Address Visibility	Direct Translation(3)
BRAF	BRAF	Block Repeat Active Flag	Direct Translation(3)
C	C	Carry	Direct Translation
CAR1	-	Circular Buffer Aux. Register 1	Invalid Field(3)
CAR2	-	Circular Buffer Aux. Register 2	Invalid Field(3)
CENB1	-	Circular Buffer Enable 1	Invalid Field(3)
CENB2	-	Circular Buffer Enable 2	Invalid Field(3)
CNF	-	On-chip RAM configuration	Invalid Field
DP	DP	Data Page pointer	Direct Translation(2) (3)
HM	HM	Hold Mode bit	Direct Translation
INTM	INTM	Global Interrupt Mask bit	Direct Translation
IPTR	IPTR	Interrupt vector table pointer	Direct Translation(1) (3)
MPNMC	MPNMC	Microprocessor/Microcontroller	Direct Translation(3)
NDX	-	Enable INDX register	Invalid Field(3)
OV	OVA	Overflow flag	(2)
OVLY	OVLY	Internal RAM Overlay	Direct Translation(3)
OVM	OVM	Overflow mode	Direct Translation
PM	FRCT	Product Mode	PM = 0,1 only
RAM	-	Program RAM enable	Invalid Field(3)
SXM	SXM	Sign Extension Mode	Direct Translation
TC	TC	Test Control Bit	Direct Translation
TRM	-	Enable Multiple T registers	Invalid Field(3)
XF	XF	External Flag	Direct Translation

(1) Bit positions and/or functionality change between 'C5x and 'C54x
(2) Register not referenced directly in assembler but is implicit for specific instructions
(3) User defined field in assembly (using .set or .equ).

Table 6: User-defined fields in 'C5x assembly

'C5x name	'C5x Field Assembly Descriptor
AVIS	.set (PMST >> 7)&1
BRAF	.set (PMST >> 0)&1
CAR1	.set (CBCR >> 0)&7
CAR2	.set (CBCR >> 4)&7
CENB1	.set (CBCR >> 3)&1
CENB2	.equ (CBCR >> 7)&1
IPTR	.set (PMST >> 11)&1fh
MPNMC	.set (PMST >> 3)&1
NDX	.set (PMST >> 2)&1
OVLY	.set (PMST >> 5)&1
RAM	.set (PMST >> 4)&1
TRM	.set (PMST >> 1)&1

LEADT will NOT replace 'C5x shift values with the correct 'C54x equivalents in expressions such as shown above. It will however scan expressions for invalid registers (i.e. CBCR) and flag the expression as an error.

Table 7: Unused 'C54x Bit Fields

'C54x	'C54x Field Description
ASM	Accumulator shift mode
C16	Dual 16bit ALU arithmetic mode
CLKOFF	Disable CLKOUT bit
CMPT	ARP compatibility
CPL	Compiler Mode
DROM	Data ROM enable
OVB	Overflow Flag for B Accumulator
SMUL	Saturation on multiplication
SST	Saturation on Store

3.4 Data, Program, and I/O Addressing Modes

Table 8: Data, Program, and I/O addressing modes mapping

'C5x	'C54x	'C5x Addressing Mode	LEADT Treatment
dma	dma	Direct	Direct Translation(1)
dma,shift	dma,shift	Direct with shift	Direct Translation(1)
*	*AR[ARP]	Indirect	Direct Translation(4)
*+	*AR[ARP]+	Indirect with increment modify	Direct Translation(4)
*-	*AR[ARP]-	Indirect with decrement modify	Direct Translation(4)
*0+	*AR[ARP]+0	Indirect with index modify	Direct Translation(4) (6)
*0-	*AR[ARP]-0	Indirect with index modify	Direct Translation(4) (6)
*BR0+	*AR[ARP]+0B	Indirect with bit-reverse modify	Direct Translation(4) (6)
*BR0-	*AR[ARP]-0B	Indirect with bit reverse modify	Direct Translation(4) (6)
*,shift	*AR[ARP],shift	Indirect with shift	Direct Translation(4)
*,ARP	*AR[ARP]	Indirect with ARP modify	Direct Translation(4) (5)
*,shift,ARP	*AR[ARP],shift	Indirect with shift and ARP modify	Direct Translation(4) (5)
#k	#k	Short Immediate (8,9,13bit)	Direct Translation
#lk	#lk	Long Immediate (16bit)	Direct Translation
MMR	MMR	Memory mapped register	Direct Translation(7)
Dmad	Dmad	Data Memory address	Direct Translation
[ACC]	[A]	Accumulator Program addressing	Direct Translation(3)
Pmad	Pmad	Program Memory address	Direct Translation
PAx.	PA	Port Address	Direct Translation(2)

(1) Direct addressing using DP only (CPL=0)
(2) Not translated if used with memory mapped access, only with IN or OUT instructions
(3) Using instructions such as TBLR/TBLW/BACC/CALA.
(4) CMPT = 0 assumed always
(5) ARP change is recorded and used by LEADT for next in-line indirect addressing operand.
(6) Uses AR0 only as index register.
(7) Provided Memory mapped register exists on target.

Table 9: Unused 'C54x Data and Program Addressing modes

'C54x	'C54x Addressing Description
*SP()	Stack pointer relative
*+ARx.	Indirect with Pre-incrementation
*ARx-%	Indirect with modulo circular addressing
*ARx+%	Indirect with modulo circular addressing
*ARx-0%	Indirect with modulo circular addressing and offset
*ARx+0%	Indirect with modulo circular addressing and offset
*ARx(lk)	Indirect with long immediate offset and no modify
*+ARx(lk)	Indirect with long immediate offset and pre-modify
*+ARx(lk)%	Indirect with long immediate offset, pre-modify, and circular
Xmem,Ymem	1 or 2 indirect operands per instruction
*(lk)	Absolute addressing
[XPC]	Far program addressing (i.e. 23bit)

3.5 Conditional Code Mapping

The combinations of conditional codes that can be used on the 'C5x are more flexible than that of the 'C54x (i.e. mixing control and signed conditions). If this 'C54x criteria is breached. then LEADT will flag it with an error.

Table 10: Conditional Code Mapping

'C5x	'C54x	'C5x Condition Description	LEADT Treatment
EQ	AEQ	ACC = 0	Direct Translation
NEQ	ANEQ	ACC <> 0	Direct Translation
LT	ALT	ACC < 0	Direct Translation
LEQ	ALEQ	ACC <= 0	Direct Translation
GT	AGT	ACC > 0	Direct Translation
GEQ	AGEQ	ACC => 0	Direct Translation
C	C	Carry = 1	Direct Translation
NC	NC	Carry = 0	Direct Translation
OV	AOV	Overflow detected	Direct Translation
NOV	ANOV	No Overflow detected	Direct Translation
BIO	BIO	BIO signal Low	Direct Translation
TC	TC	Test Control = 1	Direct Translation
NTC	NTC	Test Control = 0	Direct Translation
UNC	UNC	Unconditional	Direct Translation

Table 11: Unused 'C54x Conditional Codes

'C54x	'C54x Condition Description
BOV	Overflow detected (B)
BNOV	No Overflow detected (B)
BEQ	B = 0
BNEQ	B <> 0
BLT	B < 0
BLEQ	B <= 0
BGT	B > 0
BGEQ	B => 0
NBIO	BIO signal High

3.6 Assembly Directive Mapping

For directives that are not mentioned, Direct Translation with no change applies.

Table 12: Assembly Directive Mapping

'C5x	'C54x	'C5x Condition Description	LEADT Treatment
.version [50-57]	.version 540	Device version specifier	Direct Translation
.mmregs	.mmregs	Memory mapped register definition	Direct Translation(1)
.macro	.macro	Macro definition	Direct Translation(2)
.set, .equ	.set, .equ	Set directive	Direct Translation(3)
[default]	[default]	All other directives	Direct Translation

(1) If .mmregs is not found in 'C5x source, it is added by LEADT
(2) If .macro is found then macro name is treated as instruction and copied without change to output.
(3) Expressions are scanned for invalid registers or fields.

4. Instruction Mapping

This section describes the instruction or mnemonic mapping used when performing the translation between the 'C5x and 'C54x. The process is similar to the architecture mapping where caveats are marked with superscript and «Direct Translation» means that LEADT will map as per the original and target objects listed in the table.

To achieve bit accuracy with the 'C5x it is sometimes necessary to translate one 'C5x instruction to two or three for the 'C54x. This is called an «Instruction Combination» and is marked accordingly in the mapping tables. A description of the instruction combinations that are used is given in chapter 5. In all cases, a **maximum** of 3 instructions is all that is necessary; when more are required then that instruction is deemed to be untranslatable (3 instructions does not include NOP's). It is possible that all 'C5x instructions could be emulated with 'C54x equivalents by saving and restoring program context. LEADT **does not** do this.

Another table is included within section detailing the 'C54x instructions and descriptions that LEADT cannot or does not use. This is useful information in the translation process for the user when understanding the 'C54x resources available for optimising the code further.

Acronyms that are used in the following tables are described below.

Table 13: Operand Acronyms

Operand	Description
A	C54x A register
ACC	'C5x Accumulator
ACCB	'C5x Accumulator Buffer
ARP	Auxiliary Register Pointer
ARx	Auxiliary Register 0-7
addr	Address constant
B	'C54x B register
conds	Conditional codes
dma	Direct or Indirect data memory addressing
k	Short Immediate
lk	Long Immediate
mmr	Memory mapped register (i.e. DP = 0)
pma	Program memory address
shift	Long Constant shift operand
shf	Short Constant shift operand

4.1 Accumulator Source Instructions

Instructions that do not use Data or Program memory as the source of the operation. The accumulators are the only source.

Table 14: Accumulator Source Instructions

'C5x	'C54x	'C5x Instruction Description	LEADT Treatment
ABS	ABS A	Absolute value of ACC	Direct Translation
ADCB	ADDC/ADD	ACC = ACC + ACCB + C	Direct Translation(2)
ADDB	ADD A,B,A	ACC = ACC + ACCB	Direct Translation
ANDB	AND A,B,A	ACC = ACCB & ACC	Direct Translation
BSAR k (k=1...16)	LD A,-k, A	ACC >> k (barrel shift)	Direct Translation(2)
CMPL	CMPL A	Complement ACC	Direct Translation
CRGT	MAX A	ACC = Max(ACC,ACCB), set C	Direct Translation(2)
CRLT	MIN A	ACC = min(ACC,ACCB), set C	Direct Translation(2)
EXAR	-	Exchange ACCB and ACC	(3)
LACB	LD B,A	ACC = ACCB	Direct Translation
NEG	NEG A	Negate ACC	Direct Translation
NORM *[+/-]	EXP A,NORM A	Normalise ACC	Direct Translation(2)
ORB	OR A,B,A	OR ACCB with ACC	Direct Translation
ROL	ROL A	Rotate ACC << 1	Direct Translation
ROLB	ROL, ROL	Rotate ACCB and ACC << 1	Direct Translation(2)
ROR	ROR A	Rotate ACC >> 1	Direct Translation
RORB	ROR, ROR	Rotate [ACCB | ACC] >> 1	Direct Translation(2)
SACB	LD A,B	ACCB = ACC	Direct Translation
SATH	-	ACC >> 16 if T[4:4] = 1	(4) (5)
SATL	-	ACCL >> T[3:0]	(4) (5)
SBB	SUB B,A	ACC = ACC - ACCB	Direct Translation
SBBB	SUBB/SUB	ACC = ACC - ACCB -B	Direct Translation(2)
SFL	SFTL A,1	ACC << 1	Direct Translation
SFLB	SFTL, ROL	[ACC | ACCB] << 1	Direct Translation(2)
SFR	SFTA, SFTA	ACC >> 1	Direct Translation(2)
SFRB	SFTA,SFTA,ROL	[ACC | ACCB] >> 1	Direct Translation(2)
XORB	XOR B,A	ACC = ACCB XOR ACC	Direct Translation
ZAP	LD #0, A	ACC = P = 0	Direct Translation(1)

(1) PREG is invalid register for target.
(2) Instruction combination - See chapter 5
(3) Swap register names (A and B) explicitly in assembly.
(4) Need to negate TREG, and use NORM A
(5) TREG1 not available on target, T used instead.

Table 15: Unused 'C54x Accumulator Source Instructions

'C54x	'C54x Instruction Description
ADD [ASM]	Add with fixed shift or using ASM register
AND	AND with fixed shift
LD [ASM]	Load with fixed shift or using ASM register
OR	OR with fixed shift
RND	Round accumulator (2^{15})
ROLTC	Rotate register left with TC shifted into LSB
SAT	Saturate accumulator
SFTC	Shift Register left if 2 sign bits
SUB [ASM]	Sub with fixed shift or using ASM register
XOR	XOR with fixed shift

4.2 Accumulator and Memory Source Instructions

Instructions that use Accumulators, Program and Data memory as sources.

Table 16: Accumulator and Memory Source Instructions

'C5x	'C54x	'C5x Instruction Description	LEADT Treatment
ADD dma[,shift]	ADD dma[,shift],A	ACC +=dma [<< shift]	Direct Translation
ADD #k	ADD #lk,A	ACC +=lk	Direct Translation
ADD #lk[,shift]	ADD #lk[,shift],A	ACC +=lk [<< shift]	Direct Translation
ADD dma,16	ADD dma,16,A	ACC +=dma << 16	Direct Translation
ADDC dma	ADDC dma,A	ACC +=(dma + C)	Direct Translation
ADDS dma	ADDS dma,A	ACC +=(unsigned)dma	Direct Translation
ADDT dma	ADDT dma,TS,A	ACC +=dma << T	Direct Translation(2)
AND dma	AND dma,A	ACC =dma & ACC	Direct Translation
AND #lk[,shift]	AND #lk[,shift],A	ACC =lk[,<<shift] & ACC	Direct Translation
AND #lk,16	AND #lk,16,A	ACC =lk<<16 | ACC	Direct Translation
LACC dma[,shift]	LD dma[,shift],A	ACC =dma [<< shift]	Direct Translation
LACC #lk[,shift]	LD #lk[,shift],A	ACC =lk [<< shift]	Direct Translation
LACC dma,16	LD dma,16,A	ACC =dma << 16	Direct Translation
LACL #k	LD #k, A	ACCL =k	Direct Translation(1)
LACL dma	LDU dma,A	ACCL =(dma + C)	Direct Translation
LACT dma	LD dma,TS,A	ACC =dma << T	Direct Translation(2)
LAMM mmr	LDM mmr,A	ACC =mmr	Direct Translation
OR dma	OR dma,A	ACC =dma | ACC	Direct Translation
OR #lk[,shift]	OR #lk[,shift],A	ACC =lk[,<<shift] | ACC	Direct Translation
OR #lk,16	OR #lk,16	ACC =lk<<16 | ACC	Direct Translation
SACH dma[,shf]	STH A,dma[,shift]	dma = ACCH<<shf	Direct Translation
SACL dma[,shf]	STL A,dma[,shift]	dma = ACCL<<shf	Direct Translation
SAMM mmr	STLM A,mmr	mmr = ACCL	Direct Translation
SUB dma[,shift]	SUB dma[,shift],A	ACC -=dma [<< shift]	Direct Translation
SUB #k	SUB #lk,A	ACC -=lk	Direct Translation
SUB #lk[,shift]	SUB #lk[,shift],A	ACC -=lk [<< shift]	Direct Translation
SUB dma,16	SUB dma,16,A	ACC -=dma << 16	Direct Translation
SUBB dma	SUBB dma,A	ACC -=(dma + !C)	Direct Translation
SUBC dma	SUBC dma,A	ACC -=dma (conditional)	Direct Translation
SUBS dma	SUBS dma,A	ACC -=(unsigned)dma	Direct Translation
SUBT dma	SUB dma,TS,A	ACC -=dma << T	Direct Translation(2)
XOR dma	XOR dma,A	ACC =dma XOR ACC	Direct Translation
XOR #lk[,shift]	XOR #lk[,shift],A	ACC =lk[,<<shift] XOR ACC	Direct Translation
XOR #lk,16	XOR #lk,16,A	ACC =lk<<16 XOR ACC	Direct Translation
ZALR dma	LDR dma, A	ACC=0, ACCH=dma	Direct Translation

(1) LD will not sign extend for range of k (-1 < k < 256)

(2) TREG1 not available on target, T used instead.

Table 17: Unused 'C54x Accumulator and Memory Source Instructions

'C54x	'C54x Instruction Description
ABDST Xmem,Ymem	ABS distance of 2 memory values
ADD Xmem,Ymem	Add 2 data memory operands
BIT	Test bit in memory location
CMPS	Compare, Select, Store
DADD	32bit add with memory
DADST	32bit add/sub with T
DLD	Load 32bit value
DRSUB	32bit reverse sub with memory
DSADT	32bit sub/add with T
DST	32bit store to memory
DSUB	32bit sub with memory
DSUBT	32bit sub with T
LD (ASM)	Load with ASM shift
SACCD	Conditional store of accumulator
SRCCD	Conditional store of BRC
ST || LD	Parallel store, load
ST || ADD	Parallel store, add
ST || SUB	Parallel store, sub
STH/L (ASM)	Store with ASM shift
STRCD	Conditional store of T
SUB Xmem,Ymem	Sub 2 data memory operands

4.3 Auxiliary Register and Data Page Pointer Instructions

Table 18: Auxiliary Register and Data Page Pointer Instructions

'C5x	'C54x	'C5x Instruction Description	LEADT Treatment
ADRK k	MAR *+AR[ARP](+k)	Add short immediate to AR[ARP]	Direct Translation
CMPR [0,1,2,3]	CMPR #[0,1,2,3],AR[ARP]	Compare AR[ARP] with ARCR	Direct Translation(1)
LAR ARx,dma	MVDK dma,ARx	Load ARx. from memory	Direct Translation(3)
LAR ARx,#k	STM #k, ARx	Load ARx. With short immed.	Direct Translation
LAR ARx,#lk	STM #lk, ARx	Load ARx. With long immed.	Direct Translation
LDP dma	LD dma, DP	Load DP from memory	Direct Translation
LDP #k	LD #k, DP	Load DP with 9bit immediate	Direct Translation
MAR *,ARP	-	Modify ARP	Direct Translation(2)
MAR *[+/-][,ARP]	MAR *AR[ARP][+/-]	Modify auxiliary register	Direct Translation
SAR ARx,dma	MVKD ARx,dma	Store Auxiliary register to mem.	Direct Translation
SBRK k	MAR *+AR[ARP](-k)	Sub short immediate from AR[ARP]	Direct Translation

(1) AR0 is used as compare register for 'C54x. User must initialise manually.
(2) Instruction deleted but ARP kept for future use.
(3) MVDK used instead of MVDM to avoid latency problems.

4.4 Parallel Logic Unit Instructions

Table 19: Parallel Logic Unit Instructions

'C5x	'C54x	'C5x Instruction Description	LEADT Treatment
APL dma	-	AND DBMR with dma	(1)
APL #lk,dma	ANDM #lk,dma	AND Long Immediate with dma	Direct Translation(2)
CPL dma	-	Compare DBMR with dma	(1)
CPL #lk,dma	CMPM #lk,dma	Compare Long Immediate with dma	Direct Translation(2)
OPL dma	-	OR DBMR with dma	(1)
OPL #lk,dma	ORM #lk,dma	OR Long Immediate with dma	Direct Translation(2)
SPLK #lk, dma	ST #lk, dma	Store Long Immediate to dma	Direct Translation
XPL dma	-	EXOR DBMR with dma	(1)
XPL #lk,dma	XORM #lk,dma	EXOR Long Immediate with dma	Direct Translation(2)

(1) DBMR is invalid register for target.
(2) TC bit is not affected by this instruction on target

4.5 T Register, P Register, and Multiply Instructions

Table 20: T Register, P Register, and Multiply Instructions

'C5x	'C54x	'C5x Instruction Description	LEADT Treatment
APAC	-	Add PREG to ACC	(1) (2)
LPH dma	-	Load PREG from data mem	(1)
LT dma	LD dma, T	Load T from data mem	Direct Translation
LTA dma	LD dma, T	Load T from data mem & ACC +=P	Direct Translation(1) (2)
LTD dma	LD dma, T	Load T from data mem & ACC +=P & dmov	Direct Translation(1) (2)
LTP dma	LD dma, T	Load T from data mem & ACC=P	Direct Translation(1) (2)
LTS dma	LD dma, T	Load T from data mem & ACC-=P	Direct Translation(1) (2)
MAC pma, dma	MACP dma,pma,A	Multiply & accumulate	Direct Translation(3)
MACD pma,dma	MACD dma,pma,A	Multiply & accumulate & dmov	Direct Translation(3)
MADD dma	-	Multiply & accumulate & dmov using BMAR	(4)
MADS dma	-	Multiply & accumulate & dmov using BMAR	(4)
MPY dma	MPY dma,A	Multiply signed	Direct Translation(2) (3)
MPY #k	MPY #lk,A	Multiply signed with short immed.	Direct Translation(2) (3)
MPY #lk	MPY #lk,A	Multiply signed with long immed.	Direct Translation(2) (3)
MPYA dma	MPY dma,A	Multiply & accumulate	Direct Translation(3)
MPYS dma	MPY dma,A	Multiply & accumulate	Direct Translation(3)
MPYU dma	MPYU dma ,A	Multiply unsigned	Direct Translation(2) (3)
PAC	-	ACC = PREG	(1) (2)
SPAC	-	ACC -=PREG	(1) (2)
SPH dma	-	Store PREG hi to data mem.	(1)
SPL dma	-	Store PREG lo to data mem.	(1)
SPM 0	RSBX FRCT	PREG shift count = 0	Direct Translation(1)
SPM 1	SSBX FRCT	PREG shift count = 1	Direct Translation(1)
SPM [2,3]	-	PREG shift count = 4,-6	(5)
SQRA dma	SQUR dma,A	Square & accumulate	Direct Translation(3)
SQRS dma	SQUR dma,A	Square & accumulate	Direct Translation(3)
ZPR	-	Zero PREG	(1)

(1) PREG is invalid register for target.
(2) Instruction combination - See chapter 5
(3) Non-pipelined multiply on target
(4) BMAR is invalid register for target.
(5) Use guard bands (AG/BG) on target instead

Table 21: Unused 'C54x T Register, P Register, and Multiply Instructions

'C54x	'C54x Instruction Description
FIRS	Symmetrical filter operation
LD || MAC	Load and MAC
LD || MACR	Load and MACR
LD || MAS	Load and MAS
LD || MASR	Load and MASR
LMS	Least mean squares filter
MACR	Multiply/accumulate with rounding
MACA[R]	Multiply/accumulate with AH as input [& rounding]
MACSU	Multiply/accumulate signed/unsigned
MASR	Multiply/accumulate with rounding
MASA[R]	Multiply/accumulate with AH as input [& rounding]
MPYA	Multiply with AH as input
MPYR	Multiply & rounding
POLY	Polynomial operation
SQDST	Square distance
SQUR	Square with A register input
SQUR[A,S]	Square with Accumulate
ST || MAC	Store and MAC
ST || MACR	Store and MACR
ST || MAS	Store and MAS
ST || MASR	Store and MASR
ST || MPY	Store and MPY

4.6 Branch, Call, Return Instructions

Table 22: Branch, Call, Return Instructions

'C5x	'C54x	'C5x Instruction Description	LEADT Treatment
B pma	B pma	Branch	Direct Translation
B pma, *[+/-][,ARP]	BD pma	Branch with AR update	Direct Translation(2)
BACC	BACC	Branch on ACC	Direct Translation
BACCD	BACCD	Delayed Branch on ACC	Direct Translation
BANZ pma	BANZ pma,*ARx-	ARx Conditional Branch	Direct Translation
BANZ pma,*[+/-],ARP	BANZ pma,*ARx[+/-]	ARx Conditional Branch	Direct Translation
BANZD pma	BANZ pma,*ARx-	ARx Conditional delayed Branch	Direct Translation
BANZD pma,*[+/-],ARP	BANZ pma,*ARx[+/-]	ARx Conditional delayed Branch	Direct Translation
BCND pma,conds	BC pma,conds	Conditional Branch	Direct Translation(4)
BCNDD pma,conds	BCD pma,conds	Conditional delayed Branch	Direct Translation(4)
BD pma	BD pma	Delayed Branch	Direct Translation
BD pma[,*,ARP]	BD pma	Delayed Branch with AR update	Direct Translation(2)
CALA	CALA A	Call on ACC	Direct Translation(1)
CALAD	CALAD A	Delayed Call on ACC	Direct Translation(1)
CALL pma	CALL pma	Call	Direct Translation(1)
CALL pma*[+/-][,ARP]	CALLD pma	Call with AR update	Direct Translation(1) (2)
CALLD pma	CALLD pma	Delayed Call	Direct Translation(1)
CALLD pma[,*,ARP]	CALLD pma	Delayed Call with AR update	Direct Translation(1) (2)
CC pma,conds	CC pma,conds	Conditional Call	Direct Translation(1) (4)
CCD pma,conds	CCD pma,conds	Conditional Delayed Call	Direct Translation(1) (4)
INTR k	INTR k+15	Software Interrupt	Direct Translation(1) (5)
NMI	INTR 1	Non-maskable Interrupt	Direct Translation(1)
RET	RET	Return	Direct Translation(1)
RETC conds	RC conds	Conditional Return	Direct Translation(1) (4)
RETD	RETD	Delayed return	Direct Translation(1)
RETE	RETE	Return from interrupt with enable	Direct Translation(1) (6)
RETI	RET	Return from interrupt	Direct Translation(1) (6)
TRAP	TRAP 2	Software TRAP	Direct Translation(1)

(1) Stack for 'C54x is software and must have SP initialised
(2) Requires Instruction combination - see chapter 5
(4) Conditional checking is performed.
(5) Interrupt vector changes - see chapter 5
(6) No shadow registers on target

Table 23: Unused 'C54x Branch, Call, Return Instructions

'C54x	'C54x Instruction Description
FB[D]	Far Branch (548/9)
FBACC[D]	Far Branch on ACC(548/9)
FCALA[D]	Far Call on ACC(548/9)
FCALL[D]	Far Call (548/9)
FRET[D]	Far Return (548/9)
FRETE[D]	Far Return with INTM enable (548/9)
RESET	Software Reset
RETF[D]	Fast Return

4.7 Program Control Instructions

Instructions that modify the Program Counter in a non-consecutive manner, not including Branch instructions

Table 24: Program Control Instructions

'C5x	'C54x	'C5x Instruction Description	LEADT Treatment
POP	POPM AL	Pop low ACC from stack	Instruction Combo(1) (2)
POPD dma	POPD dma	Pop Data memory from stack	Direct Translation(1)
PSHD dma	PSHD dma	Push Data memory to stack	Direct Translation(1)
PUSH	PUSHM AL	Push Low ACC to stack	Direct Translation(1)
RPT #k	RPT #k	Single Repeat(short Imm.)	Direct Translation
RPT #lk	RPT #lk	Single Repeat(long Imm.)	Direct Translation
RPT dma	RPT dma	Single Repeat(dma)	Direct Translation
RPTB	RPTB	Block Repeat	Direct Translation
RPTZ #lk	RPTZ A,#lk	Single Repeat with ACC clear	Direct Translation
XC 1,conds	XC 1,conds	Execute conditional 1	Direct Translation(2) (3)
XC 2,conds	XC 2,conds	Execute conditional 2	Direct Translation(2) (3)

(1) Stack for 'C54x is RAM based and must have SP initialised
(2) Requires Instruction combination - see chapter 5
(3) Conditional checking is performed.

Table 25: Unused 'C54x Program Control Instructions

'C54x	'C54x Instruction Description
FRAME	Modify stack pointer by immediate

4.8 I/O and Data Memory Operations

Table 26: I/O and Data Memory Operations

'C5x	'C54x	'C5x Instruction Description	LEADT Treatment
BLDD #addr,dma	MVKD #addr, dma	Data to data	Direct Translation
BLDD dma,#addr	MVDK dma, #addr	Data to data reversed	Direct Translation
BLDD BMAR,dma	-	Data to data using BMAR	(1)
BLDD dma,BMAR	-	Data to data using BMAR reversed	(1)
BLDP dma	-	Data to program using BMAR	(1)
BLPD #pma, dma	MVPD pma, dma	Program to data	Direct Translation
BLPD BMAR,dma	-	Program to data using BMAR	(1)
DMOV dma	DELAY dma	Data move	Direct Translation
IN dma, PA	PORTR PA,dma	I/O port to data	Direct Translation
LMMR mmr,#addr	MVDM #addr,mmr	Data to memory-mapped register	Direct Translation
LMMR *,#addr	MVDM #addr,*AR[ARP]	Data to memory-mapped register	Direct Translation
OUT dma,PA	PORTW dma, PA	Data to I/O port	Direct Translation
SMMR mmr,#addr	MVMD mmr,#addr	Memory-mapped register to data	Direct Translation
SMMR *,#addr	MVMD AR[ARP],#addr	Memory-mapped register to data	Direct Translation
TBLR dma	READA dma	Program to data using ACC	Direct Translation
TBLW dma	WRITA dma	Data to program using ACC	Direct Translation

(1) BMAR is invalid register for target.

Table 27: Unused 'C54x I/O and Data Memory Operations

'C54x	'C54x Instruction Description
ADDM	Add long immediate to data mem
MVDD	Move Xmem to Ymem
MVMM	Move Mmr to Mmr

4.9 Miscellaneous Control Instructions

Table 28: Miscellaneous Control Instructions

'C5x	'C54x	'C5x Instruction Description	LEADT Treatment
BIT dma,bit	BITF dma,#1<<(15-bit)	Test Bit (immediate)	Direct Translation
BITT dma	BITT dma	Test Bit (TREG)	Direct Translation(3)
CLRC bit	RSBX bit	Clear Bit	Direct Translation
IDLE	IDLE 1	Idle	Direct Translation
IDLE2	IDLE 2	Idle 2 (low-power mode)	Direct Translation
LST #0, dma	MVDM dma, ST0	Load Status Register	Direct Translation(1)
LST #1, dma	MVDM dma, ST1	Load Status Register	Direct Translation(1)
LST #0, *[+/-…]	MVDK *ARx[+/-], ST0	Load Status Register	Direct Translation(2)
LST #1, *[+/-…]	MVDK *ARx[+/-], ST1	Load Status Register	Direct Translation(2)
NOP	NOP	No operation	Direct Translation
SETC bit	SSBX bit	Set bit	Direct Translation
SST #0,dma	MVMD ST0, dma	Store Status Register	Direct Translation(1)
SST #1,dma	MVMD ST1, dma	Store Status Register	Direct Translation(1)
SST #0,*[+/-…]	MVKD ST0,*ARx[+/-..]	Store Status Register	Direct Translation(2)
SST #1,*[+/-…]	MVKD ST1,*ARx[+/-..]	Store Status Register	Direct Translation(2)

(1) Direct Memory Addressing Only
(2) Indirect Memory Addressing Only
(3) TREG2 not available on target, T used instead.

5. Translation Specifics

5.1 Arithmetic and Logical Differences

5.1.1 Accumulators

'C5x instructions that implicitly use the main Accumulator are translated to the 'C54x with the A Accumulator as the operand.

The 8bit guard band of 'C54x is ignored and only 32bit ACC's are considered. This is demonstrated in the shifting operations where sfl -> sftl (i.e. guard band is cleared). For the sfr translation to sfta a,-1 the sxm bit must be taken into account and it is assumed that the correct arithmetic state of the accumulator has been maintained to this point with SXM = 1.

It is not possible for the translator to assume that a soft guard band is being used by the 'C5x code.

5.2 Addressing

5.2.1 Data Addressing

Direct memory addressing is translated one-to-one, keeping the variable names intact. Indirect addressing is simplified in that all explicit LARP's and MAR's that do not modify the current ARx are deleted with the current ARP value passed onto the next inline instruction that uses indirect addressing. The 'C54x uses this value in its explicit usage of the ARx registers. This passing of the ARP can cause problems if a discontinuity is encountered and therefore a comment before each branch and label is needed for the user to complete the translation.

Example :

```
'C5xx
; Current ARP = 1
com1   B       mar1         ; before each branch

; Inline ARP = 2, Disc ARP unknown
mar2:                       ; before each label

; Inline ARP = 3, Disc ARP unknown
       .endif               ; before a directive that may change
                            ; the inline nature of the code.

; Inline ARP = Undefined, Disc ARP unknown
loop                        ; if ARP has not been defined yet
```

5.2.2 Discontinuities with AR modification

B and **Call** instructions with AR modifications are coded into the equivalent delay slot version to protect the modification from interrupts and to cut down on the cycle count. On

a 'C5x this modification comes with zero overhead using the ARP modification. For 'C54x this modification needs to be explicit.

Example :

```
'C5x                                    'C5xx
; Current ARP = 3                      ; Current ARP = 3
       b       temp,*+                         BD      temp
                                               MAR     *AR3+
                                               NOP
```

For just a change on the ARP value, then a normal branch is used with appropriate comments annotated indicating a change on the ARP. The translator will use the new ARP for subsequent ARP references.

Example :

```
'C5x                                    'C5xx
; Current ARP = 3                              ; Current ARP = 3
       b       temp,AR4                        B       temp
                                               ; Current ARP = 4
```

For **CALLD** and **BD,** where the modification needs to be done and the slots are already full, then the translation is performed as in the following example. One cycle is added and the modification is performed one cycle before it should be. This should not affect the code functionality but does add a cycle and could behave differently in the presence of interrupts.

Example :

```
'C5x                                    'C5xx
; Current ARP = 5                      ; Current ARP = 5
       calld   temp,*+,AR1                     MAR     *AR5+
       nop                                     CALLD   temp
       nop                             ; Current ARP = 1
                                               NOP
                                               NOP
```

5.2.3 BANZ

BANZ has to be translated slightly differently due to the implicit nature of the 'C5x BANZ. For 'C5xx the aux. register modification needs to be done explicitly.

Example 1:

```
'C5x                                    'C5xx
; arp = 0                              ; arp = 0
       BANZ temp                               BANZ temp,*AR0-
```

Example 2:

```
'C5x                                    'C5xx
; arp = 1                              ; arp = 1
       BANZ temp,AR2                           BANZ temp,*AR1-
                                       ; arp = 2
```

5.3 Instruction Combinations

5.3.1 NORM instruction

The NORM instruction following a RPT (Normalisation procedure for 'C5x) is translated to EXP,ST,NORM combination without the RPT. The T register needs to be unused before the EXP instruction (or saved) and the ST fills the T register latency required as well as storing T to the original AR[ARP]. The RPT instruction will not appear in the output.

Example: RPTed NORM

```
'C5x                                    'C5xx
       mar     *,AR5                    ; save T before
       rpt     #16                             exp     A
       norm    *+                              st      T, *(AR5)
                                               norm    A
```

Example : non-RPTed NORM

```
'C5x                                    'C5xx
; ARP = 5                               ; ARP = 5
       norm *+                                 SFTC    A
                                               NOP                     ;latency
                                               NOP                     ;latency
                                               XC      1,NTC
                                               MAR     *AR5+
```

5.3.2 MPY and Accumulation combinations

The instruction sequence of MPY followed by an accumulation instruction (i.e. LTA,LTD,MPYA etc.), is translated to MAC and the 'C54x equivalent to the next instruction without the Accumulation operation.

Example : MPY, LTD

```
'C5x                                            'C5xx
       lacl    #0      ;Acc=0                           ld      #0,A        ;A=0
       lt      d0      ;t=d0                            ld      d0,T        ;t=d0
       mpy     c0      ;P=co.d0                         mac     c0,A        ;A+=co.d0
       ltd     d1      ;t=d1                            ltd     d1          ;t=d1
                       ;d0=d1                                               ;d0=d1
                       ;Acc+=P
```

Table 29: Summary of 'C5x and 'C54x MPY/Accumulate mappings

'C5x	'C54x Equivalent
MPY,APAC	MAC
MPY,SPAC	MAS
MPY,PAC	MPY

```
MPY,LTA       MAC,LD(T)
MPY,LTD       MAC,LTD
MPY,LTS       MAS,LD(T)
MPY,LTP       MAC,LD(T)
MPYU,PAC      MPYU
MPYU,LTP      MPYU,LD(T)
```

5.3.3 POP

For the 'C54x, the POPM instruction is used for restoring the ACC value (A register) by accessing its lower half via the memory mapped register file. Therefore the rest of the ACC will not be changed as a result of the POP, so the extra load instruction is used to reset the A register to zero before the POP occurs.

Example :

```
'C5x                          'C5xx
      pop                     ld     #0, A
                              popm   AL
```

5.3.4 Execute conditional (XC)

For the 'C54x, the XC instruction requires one more delay slot between the cycle that sets the condition and the XC instruction. This is due to an additional phase in the pipeline. LEADT adds the NOP automatically as below.

Example : XC

```
'C5x                          'C5xx
                              NOP    ; latency nop
      XC      1,TC,C          XC     1,TC,C
        <ins1>                  <ins1>
```

5.3.5 ACC and ACCB logical instructions

The 'C5x comparison instructions can be emulated with 3 instructions each. For the MAX instruction however, when A = B, the carry will be cleared as opposed to the 'C5x CRGT instruction. Therefore the user needs to change the subsequent test.

Example : CRLT

```
'C5x                          'C5xx
      crlt                    MIN    A       ; do compare
                              XORM   #800h,*(ST0); toggle carry
                              LD     A, B    ; A = B
```

Example : CRGT

```
'C5x                          'C5xx
      crgt                    MAX    A       ; do compare
                              XORM   #800h,*(ST0); toggle carry
                              LD     A, B    ; A = B
                              ;********* WARNING - CARRY=0 IF A=B
                              ;('C5X: CARRY=1)
```

5.3.6 ACC and ACCB arithmetic instructions

The 'C5x has a number of instructions involving the ACC and ACCB intended for both 32bit and 64bit arithmetic operations. These instructions need to use the A and B registers on the 'C54x which do not have the same relationship as ACC/ACCB. Memory type instructions need to be used here that do not interfere with the context of the device. So absolute addressing is used.

Example : ADCB

```
'C5x                    'C5xx
       adcb             ADDC   *(BL), A
                        ADD    *(BH),16,A
```

Example : SBBB

```
'C5x                    'C5xx
       SBBB             SUBB   *(BL), A
                        SUB    *(BH),16,A
```

Please note that in some circumstances where OVM is not set, then the arithmetic behaviour of the 'C54x may not align to the 'C5x due to the A and B register guard bands. If a result is expected to wrap-round (i.e. 0x7fff ffff -> 0x8000 0000) on the 'C5x then a check will need to be done on the 'C54x guard band values and the register contents adjusted accordingly.

5.3.7 ACC and ACCB shift instructions

Shifting operations are also subject to the same care-abouts as the arithmetic instructions. For 'C5x arithmetic shifts, b[31:31] of the accumulator should represent the sign. However for the 'C54x it is b[39:39] that is the sign bit. So the guard band needs to be cleared and b[31:31] should be sign-extended to b[39:39] of the 'C54x guard band.

Example : BSAR

```
'C5x                    'C5xx
;assume SXM=1 always    ;assume SXM=1 always
       BSAR  16         SFTA   A,8    ; b[39:39] = sign
                        SFTA   A,-8   ; guard band is sign extended
                        LD     A,-16  ;bsar shift (LD doesn't affect carry)
```

Rotate instructions using the combined ACC and ACCB can be emulated using 2 rotate instructions as shown in the following examples.

Example : RORB

```
'C5x                            'C5xx
rorb                                   ROR     A          ; Rotate A
                                       ROR     B          ; Rotate B
```

Example : ROLB

```
'C5x                            'C5xx
rolb                                   ROL     B          ; Rotate B
                                       ROL     A          ; Rotate A
```

The 'C5x SFR uses sign extension if SXM=1. Therefore the 'C54x sign bit (b[39:39]] and guard band needs to be initialised with the 'C5x b[31:31] sign value. An additional shift is used for that purpose.

Example : SFR

```
'C5x                    'C5xx
sfr                     SFTA   A,8    ; b[39:39] = sign
                        SFTA   A,-9   ; A >> 1 with sign extension
```

Shift instructions using the combined ACC and ACCB can be emulated with the single register shift followed by a rotate.

Example : SFLB

```
'C5x                    'C5xx
sflb                    SFTL   B,1    ; Shift B
                        ROL    A      ; Rotate A
```

Example : SFRB

```
'C5x                    'C5xx
sfrb                    SFTA   A,8    ; b[39:39] = sign
                        SFTA   A,-9   ; A >> 1 with sign extension
                        ROR    B      ; Rotate B
```

5.4 Structures

5.4.1 Macros

User macros are identified by their definition (i.e. .macro) and translated accordingly per the definition. Every instance of the macro that is uncovered is then replicated in the output file. The macro name is added to the instruction list as a 'special instruction'.

For macro's that are not defined in the source but are referenced with .include or .mlib, then the translator will see it as an undefined instruction. It will automatically be considered to be a macro (i.e. original source is considered to be correctly compilable) and therefore included untranslated.

5.4.2 Conditional codes sequences

For the 'C5x, the specifying of conditional codes is more liberal that the 'C54x. For example, the 'C54x will only allow either max of 3 control conditions or 2 signed (test the accumulator sign and overflow flag). The translator will check if there is a mixed ‘control’ and ‘signed’ sequence and flag this as an error.

Example :

```
'C5x
RETC   GT,TC

'C5xx
;0058  RETC    GT,TC
; *********  ERROR - CANNOT MIX CONTROL AND SIGNED CONDITIONS ON TARGET
```

5.4.3 Interrupt Vectors

The 'C54x has a different interrupt vector table format which does affect some instructions such as INTR and TRAP. Below is a table showing differences between the 'C5x and 'C54x operand ‘k’ for the above mentioned instructions.

Table 30: Interrupt k value mapping

'C5x k value	'C54x k value	'C5x k Description	LEADT Treatment
0	0	Reset	Direct Translation
1 ... 9	16 ... 24	Core and Peripheral interrupts	Direct Translation(+15)
10 ... 16	25 ... 31	Peripheral Interrupts	Direct Translation(+15)
17	2	TRAP	Direct Translation(-15)
18	1	NMI	Direct Translation(-17)
19	3	Peripheral Interrupt	Direct Translation(-16)
20 ... 31	4 ... 15	User-Defined.	Direct Translation(-16)

6. Translation Flow

6.1 Recommended Procedure for Translation

The recommended 10 step process is as follows :

1. The original 'C5x code should pass through DSPA -v50 with no errors (essential)
2. Application Test Vectors should pass through original 'C5x code without errors
3. Run source code through LEADT
4. Check for errors and change 'C5x code to minimise errors but still pass Application Test Vectors.
5. FREEZE translator output.
6. Modify application memory map to match 'C54x target device (linker and simulator command files)
7. Modify 'C54x output code to secure final architecture mapping.
8. Run original Application Test Vectors through new 'C54x code and modify code manually till no errors.
9. Analyse performance and check if application criteria are met (i.e. cycle count, memory size ...). LEADT.INF gives a brief summary of amount of code size increase/decrease to gauge the efficiency of the translation.
10. If criteria are not met then attempt to implement 'C54x special features. The instructions, registers, and modes that are unused by LEADT are summarised in the respective sections of this document.

Please note that any anomalies in the LEADT translator that you find should be reported back to Texas Instruments through either the local sales office or Product Information Centre (PIC/EPIC).

6.2 Translator Invocation

Please use leadt -? for command line options.

```
TMS320C5x to 'C54x Source Code Translator Version 0.91-960703
Copyright (c) 1993-94  Texas Instruments Incorporated

Syntax:  LEADT [-ehinqswv?] asm_file [tran_file]

Where: Options are one or more characters preceded by '-'
       -?: This help list
       -e: Turn off errors
       -h: Turn off hints
       -i: Generate Info file
       -o: Keep original code in translated file
       -n: Suppress TI file header in .TRN file
       -q: Suppress the on-screen banner
       -s: Insert original code in comment field
       -w: Turn off warnings
       -v: Specifies a version
          -v10:TMS32010  -v20:TMS32020  -v25:TMS320c25
```

```
-v50:TMS32050 [default]
```

Or file names can be entered interactively.

```
TMS320C5x to 'C54x Source Code Translator Version 0.91-960703
Copyright (c) 1993-94  Texas Instruments Incorporated

      Source file [.asm]:
```

6.2.1 Errors vs. Warnings vs. Hints

LEADT prints error and warning messages to both the output file and STDOUT. The output file (*.TRN) also contains additional warnings (alongside errors) and hints to help the user determine and remedy the problem. Below is a description of the messages produced by LEADT.

An ERROR, meaning that the instruction did not have a translation, may have a WARNING or HINT associated with it to help the user understand the nature of the error. A WARNING signifies that the instruction was translatable but there may be a difference in the resultant code that may change the outcome of the operation. A HINT is inserted to assist the user in fixing an ERROR or in optimising the code further for the target.

Table 31: LEADT Error Descriptions

Error #	Error String and Description
01	*THIS INSTRUCTION HAS NO TRANSLATION*
02	*CAN NOT TRANSLATE, PLEASE REWRITE*
03	*OPERAND AMBIGUOUS FOR TRANSLATOR*
04	*INVALID VERSION NUMBER*
05	*MORE THAN 3 CONDITIONS NOT ALLOWED*
06	*CANNOT MIX CONTROL AND SIGNED CONDITIONS ON TARGET*
07	*DBMR REGISTER NOT VALID ON TARGET*
08	*BMAR REGISTER NOT VALID ON TARGET*
09	*INVALID REGISTER FOR TARGET: <register name>*
10	*INVALID BIT FIELD FOR TARGET: <register name>*

Table 32: LEADT Warning Descriptions

Warning #	Warning String and Description
01	*UNSUPPORTED MNEMONIC, MACRO ASSUMED*
02	*AR REGISTER MIGHT BE CHANGED*
03	*OPERAND NOT TRANSLATED*
04	*LINE TRUNCATED*
05	*NARP IN RPT NOT SUPPORTED*
06	*SUBSTITUTION SYMBOL MIGHT NOT BE EVALUATED*
07	*TC IS NOT AFFECTED BY THIS INSTRUCTION*
08	*POSSIBLE LATENCY NEEDED HERE*
09	*CHECK THAT RPT IS OK !*
10	*T IS OVERWRITTEN BY EXP, SAVE BEFORE*
11	*NO PIPELINE IN MULTIPLY*
12	*T USED INSTEAD OF TREG1*
13	*CARRY=0 IF A=B (C5X: CARRY=1)*

14	*T USED INSTEAD OF TREG2*
15	*NO SHADOW REGISTERS AVAILABLE ON TARGET FOR RETI*
16	*P REGISTER NOT AVAILABLE ON TARGET*

Table 33: LEADT HINT Descriptions

HINT #	HINT String and Description
01	*Use AG/BG instead*
02	*A and B can be operated upon in the same manner*
03	*Use NORM instead with T value negated*
04	*T value needs to be negated before following is used for right shift*
05	*Use A Accumulator instead*

6.2.2 Information File (LEADT.INF)

Example Output :

```
Information file for LEADT              **** Tue Aug 19 19:26:34 1997
--------------------------
'C5x -> 'C54x translation : LEADT version - Version 1.00-970808

Input File  : validate\c5xins.asm
Output File : validate\c5xins.trn

 36 Errors
 51 Warnings
 9 Hints

 Number of Lines not requiring translation:     60

 Number of instructions:
     Input: 344   Output: 400   Percent Increase: 16

 Number of directives: 15

 Number of macros:     0

 Invalid Registers Used: BMAR PREG

 Invalid Fields Used:
```

7. Conclusion

LEADT has been designed as a tool to assist in the translation of 'C5x to 'C54x assembly code. The differences in the architectures between the processors prohibits a ‘one-to-one’ translation process. Therefore there will be an expansion of code and accordingly cycles in the 'C54x code output from LEADT which are unavoidable until optimisation can take place.

Project teams that have worked with this utility have said that “it was invaluable in that it saved a lot of time in the beginning of the translation process. It was useful as a training aid for the 'C54x processor and the translated code did not require much effort in making the code run bit exact as per the 'C5x. Of course the optimisation stage was necessary to achieve the project’s final goals.” <Satisfied Customer>

Appendix: 'C5x to 'C54x translator revision

This file denotes revision changes only. User documentation is still being completed.
NOTE:
This utility is definitely NOT free of bugs so any feedback that you may have on existing bugs or new features that would improve its productivity would be appreciated.

Release Information

19/8/97
========
Rls1.00-970819 :

- version 1.00
- translates .version directive
- included GREG,RPTC,CWSR as invalid registers
- fixed lst and sst intructions for both direct and indirect addressing
- included CAR[1,2], CENB[1,2], NDX, TRM as invalid bit fields
- scan .equ and .set expressions for invalid registers and fields
- changed PA0-PA15 to correct I/O addresses (50h-5fh)
- made PA0-PA15 invalid if not being used by IN or OUT instruction
- added translation for BACCD, INTR, NMI, RETI,ADCB,SBBB,RORB ,ROLB,SFRB, SFLB
- change TRAP to TRAP 2 instead of 3
- add offset of 15 to INTR operand for C54x vector table starting at 1
- check conditional code combinations so that they do not violate C54x rules
- fixed SBRK, ADRK for operand immediate operand
- Fixed bad translation of BLDD dma,BMAR
- Added mpy/acc combinations for more variety
- Changed MPYA to MPY
- Changed MPYS to MPY
- Changed SQRA to SQUR
- Changed SQRS to SQUR
- Fixed BSAR instruction combinations
- Changed SFR translation to be correct
- Fixed translation for AND/OR/EXOR with immediate and shift operands
- Added more to information file
- Change translation of single NORM.

1/8/97
========
Rls0.91-970801 :

- fix banz instruction translation - used *ARP instead of *ARx
- added banzd translation
- added translator version to output file
- removed addition of .end directive
- ARP comments for Calls and Branches aligned correctly

20/6/97
========
Rls0.91-970620 :
- restore label for deleted LARP's and MAR's
- aligned directives and macros with instructions
- removed 1 ARP inline comment from near directives (.if)
- change .macro label from uppercase to lowercase
- added calad instruction.
- all undefined instructions considered to be macros and left untranslated
- if ARP=undefined then print ARP = undefined
- correct translation of addk, subk
- translate mpy,ltd -> mac,ltd
- made sure all labels had ARP comment
- removed .version 500 from translated file
- tested with Rls1.16 of C54x tools
- readme.1st -> update.txt
- readme.1st = uncomplete user's guide
- distributed both SUN and PC versions.

05/9/96
========
Rls0.91-960905 :
- fix bug: sbrk/adrk immediate problem
- keep macros in source even if detected as macro
- translate PASR/PAER to RSA/REA
- translate TREG0 to T
- fix bug: got rid of ARP load after narp=AR0

Rls0.91-960806 :
- fix bug : add #lk translated to ld #lk
- added RETD, RETE, RETI
- added RETC, RETCD
- added NOV translation
- fix bug : RPT instr comments where deleted

Rls0.91-960726 :
- LARP's are treated like MAR's
- At each label the INLINE ARP is stated
- No LD of ARP for BD's and CALLD's (translated from B and CALL)
- At each B/CALL/BD/CALLD/BC/BCD/CC/CCD the ARP is stated even if changed by that instruction

Rls0.91-960716 :
- Fixed bug: BBNZ and BBZ were reversed.

Rls0.91-960703 :
- Fixed bug: with sacl/sach using shift operand
- Removed loading of ARP before each label. Added comment instead.
- Removed clearing of ARP after each label so normal AUX reg tracking is maintained.

Rls0.91-960525 :
- Release changed to 0.91
- Fixed Bug: Macd/Mac Indirect operand
- Add warnings and toggle C for MIN/MAX compatibility
- Warnings and Errors can be turned on separately.

Rls0.90-960424 :
- add .version and .title directives into output
- conditional include of .mmregs and .end directives
- change -i switch to -o for original code inclusion
- add information file output (*.inf)
- include warnings and hints count
- Fixed bug : for lacl use ldu except immediate op's.
- Delete 'mar's that don’t modify.

Rls0.90-960419 :
- add date into version number
- C5x is default translator source
- take CMPT bit compatibility out
- add -? option to command line
- add bad command line handler
- take serial port registers out of invalid reg list

Rls0.90-941101 :
- Original