CROSS-ASSEMBLER DIRECTIVES (CONCLUDED)

PAGE TITLE — **TITL**

TITL supplies title to be printed in the heading of each page of the source listing.

Syntax: [<label>] TITL '<string>' [<comment>]

RESTART SOURCE LISTING — **LIST**

LIST restores printing of the source listing.

Syntax: [<label>] LIST [<comment>]

STOP SOURCE LISTING — **UNL**

UNL halts the source listing output until the occurrence of a LIST directive.

Syntax: [<label>] UNL [<comment>]

EJECT PAGE — **PAGE**

PAGE causes the assembler to continue the source program listing on a new page. The PAGE directive is not printed in the source listing, but the line counter increments.

Syntax: [<label>] PAGE [<comment>]

INITIALIZE WORD — **DATA**

DATA places one or more values in one or more successive words of memory.

Syntax: [<label>] DATA <exp>[,<exp>] [<comment>]

INITIALIZE TEXT — **TEXT**

TEXT places one or more characters in successive words of memory.

Syntax: [<label>] TEXT [-]'<string>' [<comment>]

DEFINE ASSEMBLY-TIME CONSTANT — **EQU**

EQU assigns a value to a symbol.

Syntax: <label> EQU <exp> [<comment>]

EXTERNAL DEFINITION — **DEF**

DEF makes one or more symbols available to other programs for reference.

Syntax: [<label>] DEF <symbol>[,<symbol>] [<comment>]

EXTERNAL REFERENCE — **REF**

REF provides access to one or more symbols defined in other programs.

Syntax: [<label>] REF <symbol>[,<symbol>] [<comment>]

SECONDARY EXTERNAL REFERENCE — **SREF**

SREF provides access to one or more symbols defined in other programs.

Syntax: [<label>] SREF <symbol>[,<symbol>] [<comment>]

FORCE LOAD — **LOAD**

LOAD is similar to REF, but the symbol does not need to be used in the module containing the LOAD. The symbol used in LOAD must be defined in some other module. LOADs are used with SREFs.

Syntax: [<label>] LOAD <symbol>[,<symbol>] [<comment>]

PROGRAM END — **END**

END terminates the assembly. The last source statement of a program is the END directive.

Syntax: [<label>] END [<symbol>] [<comment>]

COPY SOURCE FILE — **COPY**

COPY changes the source input for the assembler.

Syntax: [<label>] COPY <file name> [<comment>]

DEFINE MACRO LIBRARY — **MLIB**

MLIB provides the name of a library containing macro definitions.

Syntax: [<label>] MLIB '<pathname>' [<comment>]

TMS32010
DIGITAL SIGNAL PROCESSOR
Programmer's Reference Card

ASCII REFERENCE TABLE

	00	10	20	30	40	50	60	70
00	NUL	DLE	SP	0	@	P	\	p
01	SOH	DC1	!	1	A	Q	a	q
02	STX	DC2	"	2	B	R	b	r
03	ETX	DC3	#	3	C	S	c	s
04	EQT	DC4	$	4	D	T	d	t
05	ENQ	NAK	%	5	E	U	e	u
06	ACK	SYN	&	6	F	V	f	v
07	BEL	ETB	'	7	G	W	g	w
08	BS	CAN	(	8	H	X	h	x
09	HT	EM	)	9	I	Y	i	y
0A	LF	SUB	*	:	J	Z	j	z
0B	VT	ESC	+	;	K	[	k	{
0C	FF	FS	,	<	L	\	l	|
0D	CR	GS	–	=	M	]	m	}
0E	SO	RS	.	>	N	^	n	~
0F	SI	US	/	?	O	_	o	DEL

HEX-DECIMAL TABLE

HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0000	0	000	0	00	0	0	0
1000	4,096	100	256	10	16	1	1
2000	8,192	200	512	20	32	2	2
3000	12,288	300	768	30	48	3	3
4000	16,384	400	1,024	40	64	4	4
5000	20,480	500	1,280	50	80	5	5
6000	24,576	600	1,536	60	96	6	6
7000	28,672	700	1,792	70	112	7	7
8000	32,766	800	2,048	80	128	8	8
9000	36,864	900	2,304	90	144	9	9
A000	40,960	A00	2,560	A0	160	A	10
B000	45,066	B00	2,816	B0	176	B	11
C000	49,152	C00	3,072	C0	192	C	12
D000	53,248	D00	3,328	D0	208	D	13
E000	57,344	E00	3,584	E0	224	E	14
F000	61,440	F00	3,840	F0	240	F	15

RTC HOTLINE NUMBERS

For help with the TMS32010, call the TI Regional Technology Center nearest you. The centers are staffed with applications engineers ready to answer all your questions.

Atlanta	404/452-4686
Boston	617/890-4271
Chicago	312/228-6008
Dallas	214/680-5096
Northern California	408/980-0305
Southern California	714/660-8164

TEXAS INSTRUMENTS

SYMBOLS FOR INSTRUCTION SET SUMMARY

SYMBOL	MEANING
D	Data memory address field
I	Addressing mode bit
K	Immediate operand field
PA	3-bit port address field (PA0 through PA7 are predefined assembler symbols equal to 0 through 7, respectively)
R	1-bit operand field specifying auxiliary resister
S	4-bit left-shift code
X	3-bit accumulator left-shift field

INSTRUCTION SET SUMMARY

MNEMONIC	DESCRIPTION	NO. CYCLES	NO. WORDS	OPCODE INSTRUCTION REGISTER 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ABS	Absolute value of accumulator	1	1	0 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0
ADD	Add to accumulator with shift	1	1	0 0 0 0 ← S → I ← D →
ADDH	Add to high-order accumulator bits	1	1	0 1 1 0 0 0 0 0 I ← D →
ADDS	Add to accumulator with no sign extension	1	1	0 1 1 0 0 0 0 1 I ← D →
AND	AND with accumulator	1	1	0 1 1 1 1 0 0 1 I ← D →
APAC	Add P Register to accumulator	1	1	0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1
B	Branch unconditionally	2	2	1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BANZ	Branch on auxiliary resister not zero	2	2	1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BGEZ	Branch if accumulator ≥ 0	2	2	1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BGZ	Branch if accumulator > 0	2	2	1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BIOZ	Branch on $\overline{BIO}$ = 0	2	2	1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BLEZ	Branch if accumulator ≤ 0	2	2	1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BLZ	Branch if accumulator < 0	2	2	1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BNZ	Branch if accumulator ≠ 0	2	2	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BV	Branch on overflow	2	2	1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BZ	Branch if accumulator = 0	2	2	1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
CALA	Call subroutine from accumulator	2	1	0 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0
CALL	Call subroutine immediately	2	2	1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
DINT	Disable interrupt	1	1	0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1
DMOV	Copy contents of data memory location into next location	1	1	0 1 1 0 1 0 0 1 I ← D →
EINT	Enable interrupt	1	1	0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0
IN	Input data from port	2	1	0 1 0 0 0 ← PA → I ← D →
LAC	Load accumulator with shift	1	1	0 0 1 0 ← S → I ← D →
LACK	Load accumulator immediate	1	1	0 1 1 1 1 1 1 1 0 ← K →
LAR	Load auxiliary register	1	1	0 0 1 1 1 0 0 R I ← D →
LARK	Load auxiliary register immediate	1	1	0 1 1 1 0 0 0 R ← K →
LARP	Load auxiliary register pointer immediate	1	1	0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 K
LDP	Load data memory page pointer	1	1	0 1 1 0 1 1 1 1 I ← D →
LDPK	Load data memory page pointer immediate	1	1	0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 K
LST	Load status register	1	1	0 1 1 1 1 0 1 1 I ← D →
LT	Load T Register	1	1	0 1 1 0 1 0 1 0 I ← D →
LTA	LTA combines LT and APAC into one instruction	1	1	0 1 1 0 1 1 0 0 I ← D →
LTD	LTD combines LT, APAC, and DMOV into one instruction	1	1	0 1 1 0 1 0 1 1 I ← D →
MAR	Modify auxiliary register and pointer	1	1	0 1 1 0 1 0 0 0 I ← D →
MPY	Multiply with T Register; store product in P Register	1	1	0 1 1 0 1 1 0 1 I ← D →
MPYK	Multiply T Register with immediate operand; store product in P Register	1	1	1 0 0 ← K →
NOP	No operation	1	1	0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
OR	OR with accumulator	1	1	0 1 1 1 1 0 1 0 I ← D →
OUT	Output data to port	2	1	0 1 0 0 1 ← PA → I ← D →
PAC	Load accumulator from P Register	1	1	0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0
POP	Pop stack to accumulator	2	1	0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1
PUSH	Push stack from accumulator	2	1	0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 0
RET	Return from subroutine	2	1	0 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1
ROVM	Reset overflow mode	1	1	0 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 0
SACH	Store high-order accumulator bits with shift	1	1	0 1 0 1 1 ← X → I ← D →
SACL	Store low-order accumulator bits	1	1	0 1 0 1 0 0 0 0 0 I ← D →
SAR	Store auxiliary register	1	1	0 0 1 1 0 0 0 R I ← D →
SOVM	Set overflow mode	1	1	0 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1
SPAC	Subtract P Register from accumulator	1	1	0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0
SST	Store status register	1	1	0 1 1 1 1 1 1 0 0 I ← D →
SUB	Subtract from accumulator with shift	1	1	0 0 0 1 ← S → I ← D →

(Continued)

INSTRUCTION SET SUMMARY (CONCLUDED)

MNEMONIC	DESCRIPTION	NO. CYCLES	NO. WORDS	OPCODE INSTRUCTION REGISTER 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SUBC	Conditional subtract (for divide)	1	1	0 1 1 0 0 1 0 0 I ← D →
SUBH	Subtract from high-order accumulator bits	1	1	0 1 1 0 0 0 1 0 I ← D →
SUBS	Subtract from accumulator with no sign extension	1	1	0 1 1 0 0 0 1 1 I ← D →
TBLR	Table read from program memory to data RAM	3	1	0 1 1 0 0 1 1 1 I ← D →
TBLW	Table write from data RAM to program memory	3	1	0 1 1 1 1 1 0 1 I ← D →
XOR	Exclusive OR with accumulator	1	1	0 1 1 1 1 0 0 0 I ← D →
ZAC	Zero accumulator	1	1	0 1 1 1 1 1 1 1 1 0 0 0 1 0 0 1
ZALH	Zero accumulator and load high-order bits	1	1	0 1 1 0 0 1 0 1 I ← D →
ZALS	Zero accumulator and load low-order bits with no sign extension	1	1	0 1 1 0 0 1 1 0 I ← D →

CROSS-ASSEMBLER DIRECTIVES

ABSOLUTE ORIGIN — **AORG**

AORG places a value in the location counter and defines the succeeding locations as absolute.

Syntax: [<label>] AORG [<wd-exp>] [<comment>]

RELOCATABLE ORIGIN — **RORG**

RORG places a value in the location counter and defines the succeeding locations as program relocatable.

Syntax: [<label>] RORG [<exp>] [<comment>]

DUMMY ORIGIN — **DORG**

DORG places a value in the location counter and defines the succeeding locations as a dummy section. No object code is generated in a dummy section.

Syntax: [<label>] DORG <exp> [<comment>]

BLOCK STARTING WITH SYMBOL — **BSS**

BSS first assigns the label, if present, and then advances the location counter by the value of the expression.

Syntax: [<label>] BSS <wd-exp> [<comment>]

BLOCK ENDING WITH SYMBOL — **BES**

BES first advances the location counter by the value of the expression and then assigns the label, if present.

Syntax: [<label>] BES <wd-exp> [<comment>]

DATA SEGMENT — **DSEG**

DSEG places a value in the location counter and defines succeeding locations as data relocatable.

Syntax: [<label>] DSEG [<comment>]

DATA SEGMENT END — **DEND**

DEND terminates a block of data-relocatable code by placing a value in the location counter and defining succeeding locations as program-relocatable.

Syntax: [<label>] DEND [<comment>]

COMMON SEGMENT — **CSEG**

CSEG places a value in the location counter and defines succeeding locations as common-relocatable (i.e., relocatable with respect to a common segment).

Syntax: [<label>] CSEG ['<string> [<comment>]]

COMMON SEGMENT END — **CEND**

CEND terminates the definition of a block of common-relocatable code by placing a value in the location counter and defining succeeding locations as program-relocatable.

Syntax: [<label>] CEND [<comment>]

PROGRAM SEGMENT — **PSEG**

PSEG places a value in the location counter and defines succeeding locations as program-relocatable.

Syntax: [<label>] PSEG [<comment>]

PROGRAM SEGMENT END — **PEND**

PEND places a value in the location counter and defines succeeding locations as program-relocatable. (Since PEND properly appears only in program-relocatable code, the relocation type of succeeding locations remains unchanged.)

Syntax: [<label>] PEND [<comment>]

OUTPUT OPTIONS — **OPTION**

OPTION selects several options for the assembler listing output.

Syntax: [<label>] OPTION <option-list> [<comment>]

PROGRAM IDENTIFIER — **IDT**

IDT assigns a name to the object module produced.

Syntax: [<label>] IDT '<string>' [<comment>]

(Continued)