WEITEK

WTL 3167 FLOATING-POINT COPROCESSOR

PRELIMINARY DATA
September 1988

WEITEK
2 Pm 9/1
TIME: DATE:
PRELIMINARY
TRANSMITTED BY:
[illegible]

The WEITEK WTL 3167, also known as the WEITEK ABACUS, is a high performance single-chip floating-point coprocessor for Intel's 80386 microprocessor. It is hardware and software compatible with the WEITEK 1167 coprocessor daughter board. Fully supported by a wide selection of application packages and by high-level language compilers, under DOS, UNIX System V.3, and XENIX System V.3, the WTL 3167 provides a superior floating-point accelerator for high-end PCs, workstations, industrial robots, graphics, and numeric controllers.

Contents

WTL 3167 Floating-Point Coprocessor
September 1988

WEITEK Corporation
1060 East Arques Avenue
Sunnyvale, California 94086
Telephone (408) 738-8400

Printed in the United States of America
90 89 88 6 5 4 3 2 1

WTL 3167 FLOATING-POINT COPROCESSOR

PRELIMINARY DATA
September 1988

Features

SINGLE-CHIP FLOATING-POINT COPROCESSOR

Designed for use with the Intel 80386

Fits a standard 121-pin socket, which is a superset of the Intel 80387 coprocessor socket

Pin-for-pin compatible with WEITEK 1167 coprocessor board

Upward object-code-compatible from WEITEK 1167

HIGH PERFORMANCE

5.6 single-precision megawhetstones and 1.0 single-precision megaflops in hand-coded Linpack

HIGH-LEVEL LANGUAGES

Supported by C, FORTRAN, and Pascal compilers under UNIX System V.3, XENIX System V.3, and MS-DOS real and protected mode

IEEE FORMAT

Conforms to the IEEE standard format for floating-point arithmetic in both single- and double-precision (ANSI/IEEE Standard 754–1985)

FULL FUNCTION

Add, subtract, multiply, divide, and square root

Integer-floating-point conversions

Absolute value

Compare

Transcendental functions supported by run-time libraries

Low power CMOS

Dissipates 2.0 Watts max at 25 MHz

121-pin PGA package

Description

The WTL 3167 is a high-performance single-chip floating-point coprocessor board for Intel's 80386 32-bit microprocessor. It delivers two to three times the performance of standard 32-bit numeric coprocessors. (Benchmark results are given in figure 1.)

The interface signals between the WTL 3167 and the 80386 are provided by a 121-pin socket, called the extended math coprocessor (EMC) socket, which is a superset of the 80387 socket. The WTL 3167 is pin-for-pin compatible with the WTL 1167 coprocessor daughter board.

C, FORTRAN, and Pascal compilers fully support the WTL 3167, allowing programs to be written in high-level languages. The WTL 3167 is upward object-code-compatible with the WTL 1167 coprocessor daughter board.

This document consists of three sections: the Hardware Designer's Section, the Applications Programmer's Section, and the Systems Programmer's Section.

This data sheet is complemented by two additional documents: The *WTL 3167 Hardware Designer's Guide* and the *WTL 1167 Software Designer's Guide.* Readers familiar with the *WTL 1167 Data Sheet* can simply refer to the *WTL 1167 and WTL 3167 Compatibility* section.

Benchmark	Performance
Linpack* (SP)	1.36 MFLOPS
Linpack* (DP)	.60 MFLOPS
Whetstone (SP)	5.6 MWhetstones
Whetstone (DP)	3.7 MWhetstones
*Hand-coded	

Figure 1. Benchmark results at 25 MHz

Description, continued

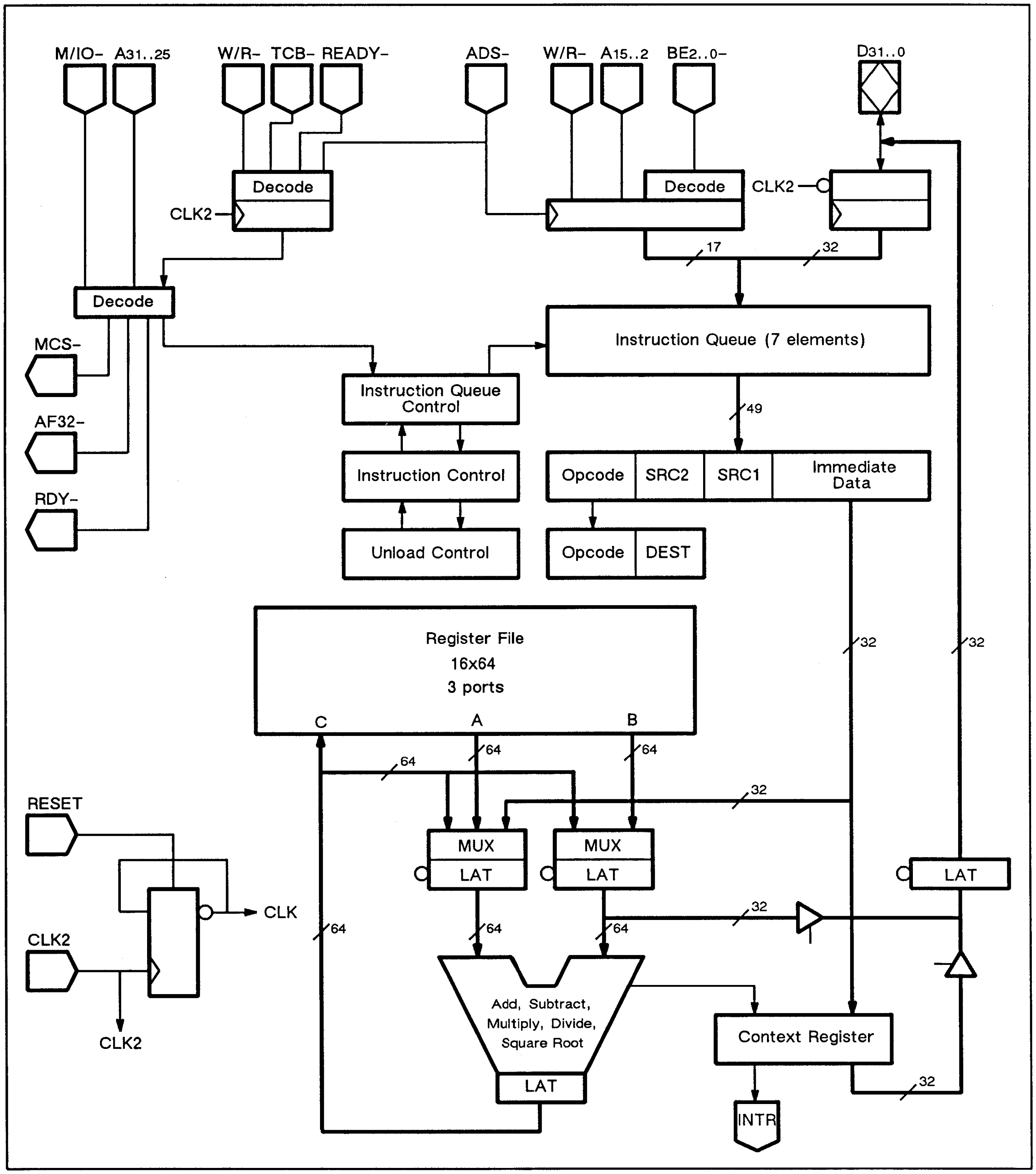


Figure 2. WTL 3167 simplified block diagram

Hardware Designer's Section

This section provides the electrical and mechanical information necessary to design the WTL 3167 into an 80386 system. For more details refer to the *WTL 3167 Hardware Designer's Guide.*

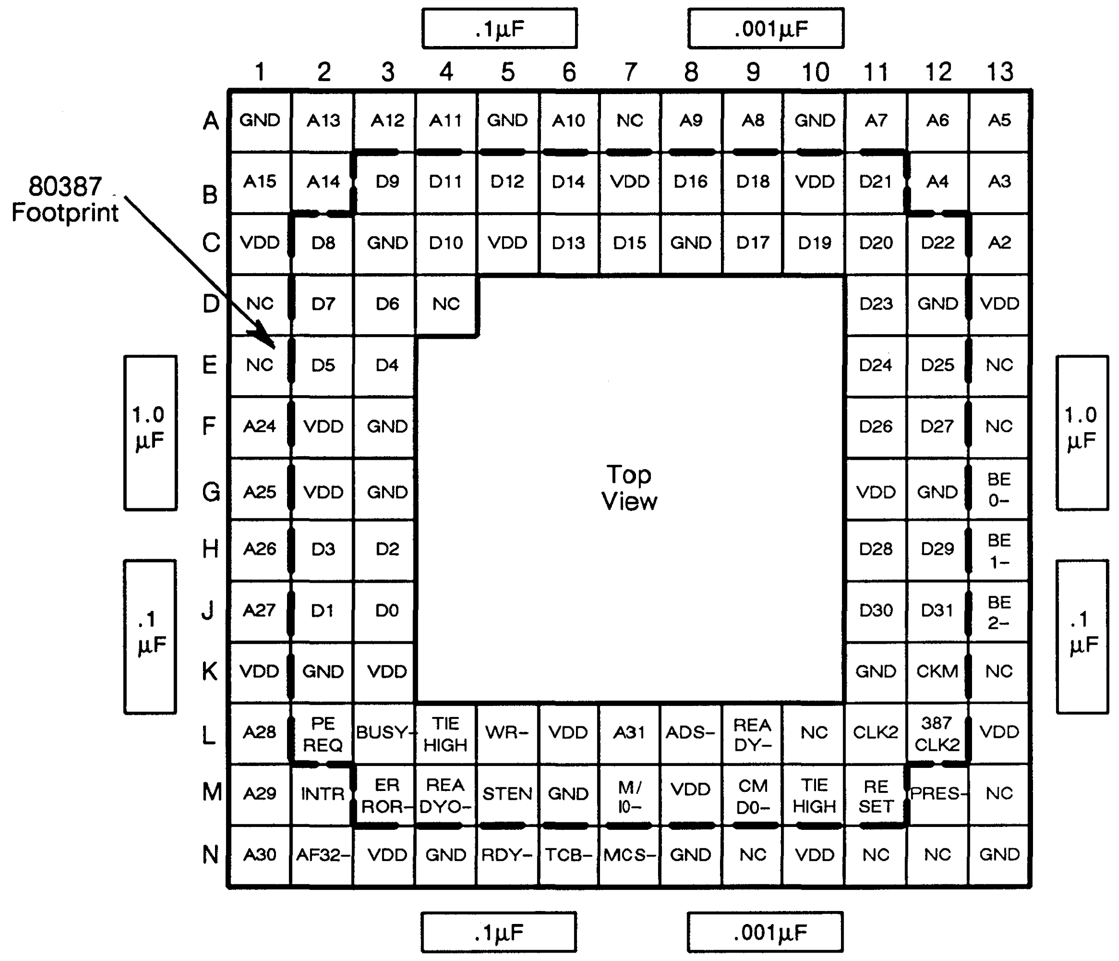
The WTL 3167 coprocessor is a memory-mapped peripheral. From the system designers standpoint, integrating the WTL 3167 into the system is as simple as adding additional memory at an upper address. To the 80386 and its application software the WTL 3167 appears to be a segment of memory. Instructions are executed by performing memory moves to and from the coprocessor.

The WTL 3167 interface to the 80386 requires signals that are not available on the 80387 socket. WEITEK has defined a superset of the 80387 socket called the *extended math coprocessor* (EMC) socket, which is a standard 121-pin pin grid array socket. With the EMC socket, a system can make use of the 80387, the WTL 3167, or both, if one uses a small daughter board that plugs into the 121-pin socket and accommodates both coprocessors. Figure 3 shows the EMC socket pinout and size.

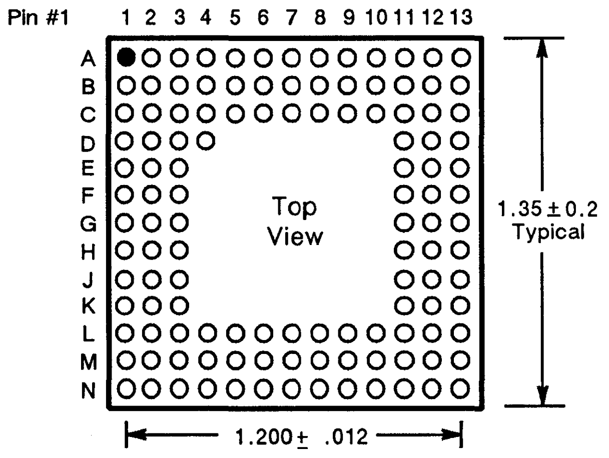
Figure 4 shows the WTL 3167 physical dimensions.

Figure 5 shows details of the WEITEK daughter board. It can accommodate both the 80387 and the WTL 3167 coprocessor.

Hardware Designer's Section, continued



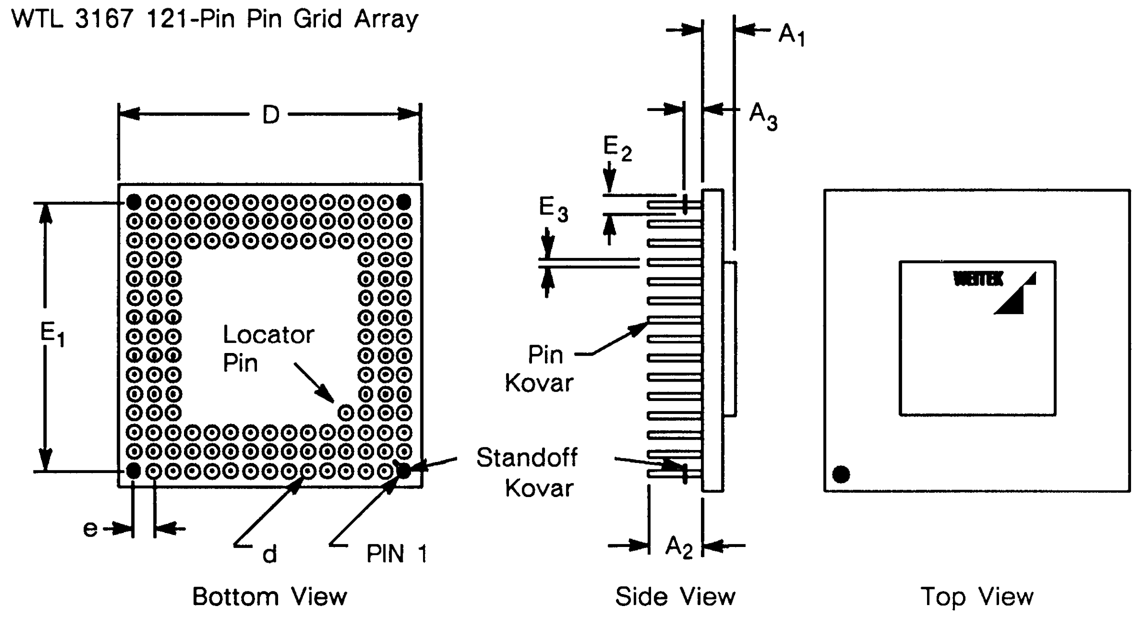
	1	2	3	4	5	6	7	8	9	10	11	12	13
A	GND	A13	A12	A11	GND	A10	NC	A9	A8	GND	A7	A6	A5
B	A15	A14	D9	D11	D12	D14	VDD	D16	D18	VDD	D21	A4	A3
C	VDD	D8	GND	D10	VDD	D13	D15	GND	D17	D19	D20	D22	A2
D	NC	D7	D6	NC							D23	GND	VDD
E	NC	D5	D4								D24	D25	NC
F	A24	VDD	GND								D26	D27	NC
G	A25	VDD	GND								VDD	GND	BE 0-
H	A26	D3	D2								D28	D29	BE 1-
J	A27	D1	D0								D30	D31	BE 2-
K	VDD	GND	VDD								GND	CKM	NC
L	A28	PE REQ	BUSY-	TIE HIGH	WR-	VDD	A31	ADS-	REA DY-	NC	CLK2	387 CLK2	VDD
M	A29	INTR	ER ROR-	REA DYO-	STEN	GND	M/ IO-	VDD	CM D0-	TIE HIGH	RE SET	PRES-	NC
N	A30	AF32-	VDD	GND	RDY-	TCB-	MCS-	GND	NC	VDD	NC	NC	GND



Recommended socket:
Robinson Nugent
P/N PGA-121AM3-S-TG
Garry
P/N 701-121-13K-LCD
Augat
P/N PGM121-1A1312-L

Figure 3. EMC socket pinout and dimensions

Hardware Designer's Section, continued



Symbol	Inches		MM	
	MAX	MIN	MAX	MIN
A1	0.135	0.080	3.43	2.03
A2	0.210	0.175	5.33	4.46
A3	0.080	0.035	2.03	0.89
D	1.400	1.280	35.56	32.51
E1	1.200 TYP		30.48 TYP	
E2	0.055	0.035	1.40	0.89
E3	0.020	0.016	0.51	0.41
d	0.075	0.035	1.91	0.89
e	0.100 TYP		2.54 TYP	

Figure 4. WTL 3167 physical dimensions

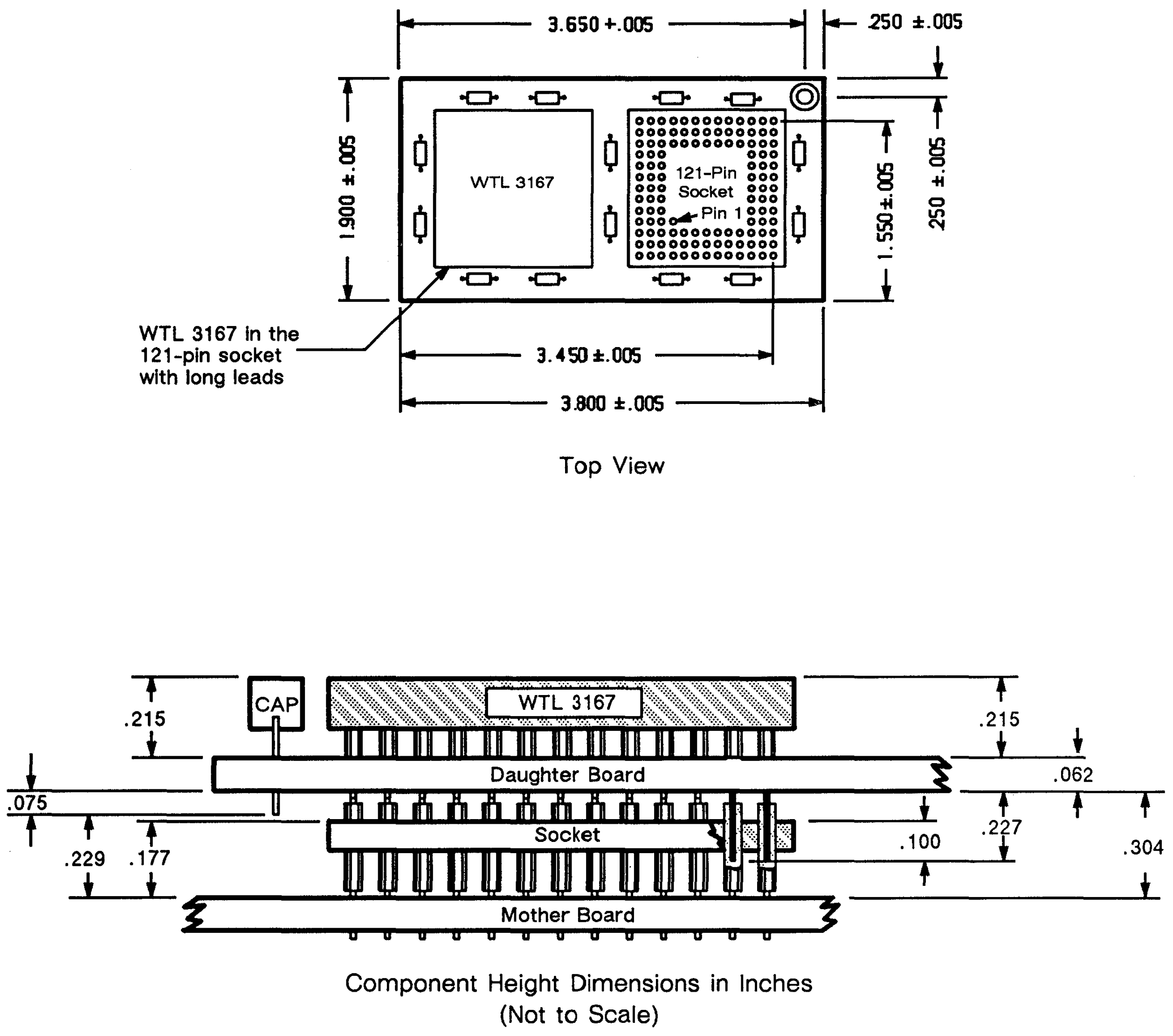


Figure 5. WTL 3167 coprocessor board physical dimensions

Hardware Designer's Section, continued

CONNECTING THE EMC SOCKET

The following paragraphs describe the connection of each EMC socket pin.

CLOCK (CLK2)

CLK2 is the clock input to the WTL 3167. All WTL 3167 timing is relative to CLK2. This signal must be the same as CLK2 of the 80386, but the WTL 3167 should have a dedicated trace. (For more details refer to the "Clock Distribution" paragraph in the *WTL 3167 Hardware Designer's Guide*).

VDD

Five volt (+5.0 V) power supply for the WTL 3167. All VDD pins must be connected.

GROUND (GND)

Ground for the WTL 3167. All ground pins must be connected.

BUSES

Address Bus ($A_{31..2}$) *and* ($BE_{2..0}$–).

Pins $A_{31..2}$ and $BE_{2..0}$– should be connected directly to the 80386 address bus and byte enables respectively.

Data Bus ($D_{31..0}$).

Pins $D_{31..0}$ should be connected to the 80386 data bus.

80386 INTERFACE SIGNALS

The 80386 interface signals are: Address Status (ADS–), Memory I/O Control Signal (M/IO–), Transfer Acknowledge (READY–), Reset (RESET), and Write/Read Line (W/R–).

ADS–, M/IO– READY–, RESET, and WR– should be connected to the 80386 ADS–, M/IO–, READY–, RESET, and W/R– respectively. Reset must be asserted synchronously with the 80386 clock to guarantee proper operation (see figure 18). In implementations using the Chips and Technologies chip set, READY– should be connected to READY– on the 82C301 and 82C302 devices, and to VCC through a 10k pull-up resistor.

COPROCESSOR READY (RDY–)

The RDY– output signal must be "ORed" into the logic generating READY– for the 80386, using only combinatorial logic (see figure 6). In implementations using Chips and Technologies chip set, RDY– should be connected to READY–.

MATH COPROCESSOR SELECT (MCS–)

MCS– is an output signal that is asserted when the current address is for the WTL 3167 coprocessor. It changes when the 80386 address bus changes. MCS– may be left unconnected or may be used in conjunction with W/R– to disable other 80386 data bus drivers on WTL 3167 read cycles. When pipelined addressing is used, special attention must be paid, as MCS– may be de-asserted prior to the end of a WTL 3167 read cycle, as shown in figure 17.

THREE CYCLE BUS (TCB–)

TCB– is an input that should be grounded in systems using the Chips and Technologies AT/386 chip set. Otherwise TCB– should be left unconnected.

AF32–

AF32– is an output signal used only in implementations based on Chips and Technologies' chip set. In implementations based on Chips and Technologies' AF32– should be connected to VCC through a 10k resistor and to AF32– on the 82C301 and 82C302 devices. It should be left unconnected otherwise.

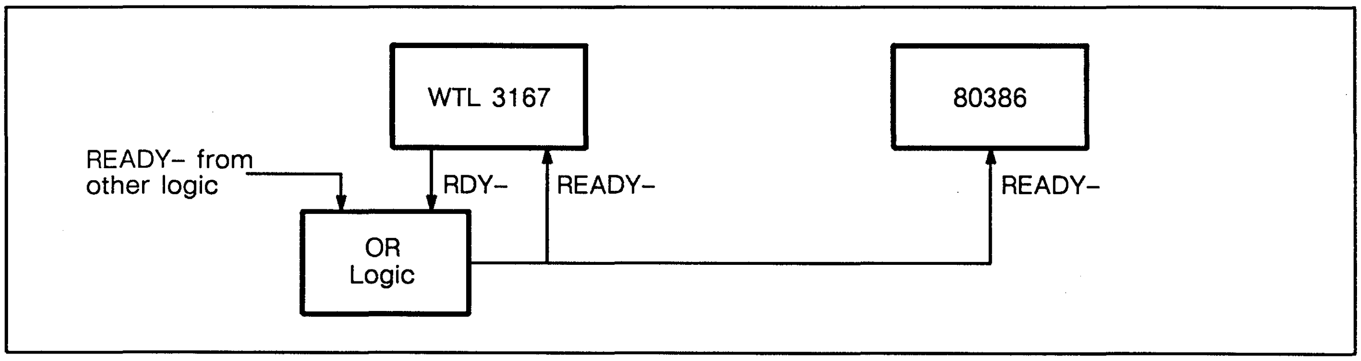


Figure 6. READY– and RDY– connection

Hardware Designer's Section, continued

INTERRUPT (INTR)

The INTR output of the WTL 3167 must be connected to the system interrupt controller. In the world of AT-compatible systems, for example, the WTL 3167 INTR should be "ORed" to the 80287/80387 interrupt logic and the output should be connected to IRQ13 as shown in figure 7.

WTL 3167 PRESENT (PRES–)

PRES– signals the presence of a WTL 3167 coprocessor. This signal should be connected to VCC through a resistor of at least 10KOhm to insure a high level when the WEITEK coprocessor is not present.

The basic software method of detecting the presence of a WTL 3167 in an 80386 system is to perform a functional test of the device by attempting to load data into the coprocessor register file and read it back (a coded example is provided in figure 56 on page 51. The hardware designer can use the PRES– output to make sure that the system generates a READY– signal when the WTL 3167 is addressed but is absent, (as determined by PRES– being high), in order to avoid system hangs.

NO CONNECTION

NC pins on the outer rows and columns are reserved for future expansion and should be left unconnected.

OTHER PINS

CKM, PEREQ, BUSY–, 387 CLK2, ERROR–, READYO–, STEN, CMD0, and pins L4 and M10 are only used by the 80387 coprocessor. Refer to the 80387 data sheet for details. Such signals can be left unconnected if the WTL 3167 is the only coprocessor that will ever be used.

DECOUPLING CAPACITORS

Decoupling capacitors should be placed on each side of the EMC socket, as shown in figure 3.

SYSTEM-LEVEL CONSIDERATIONS

The WTL 3167 coprocessor is a memory-mapped peripheral that communicates with the 80386 over the same address bus that connects the main memory to the CPU. Instructions are defined by the 14 least-significant address bits (A15..2) as well as three of the four byte enables (BE2..0–).

The seven most-significant bits of the 80386 address bus (A31..25), together with the Memory I/O control Signal (M/IO–), select the WTL 3167 coprocessor. Only the upper seven address bits are decoded to determine when a coprocessor operation is being requested.

The coprocessor will respond to memory addresses C0000000 through C1FFFFFF hex. Although by convention only addresses C0000000 to C000FFFF hex are used, it is important to be sure that other components in the system do not conflict with the address space decoded by the coprocessor. Writing to this address space will cause the WTL 3167 to execute instructions and reading will cause the coprocessor to drive the data bus.

TESTING THE DESIGN

A set of diagnostic routines that test the coprocessor design for both UNIX and DOS, real and protected mode environments, is available from WEITEK. No WTL 3167 programming knowledge is required to run the diagnostics software. Contact your WEITEK sales engineer for a copy of the diagnostics. (Refer to figure 21 on page 20 for the product's part number.)

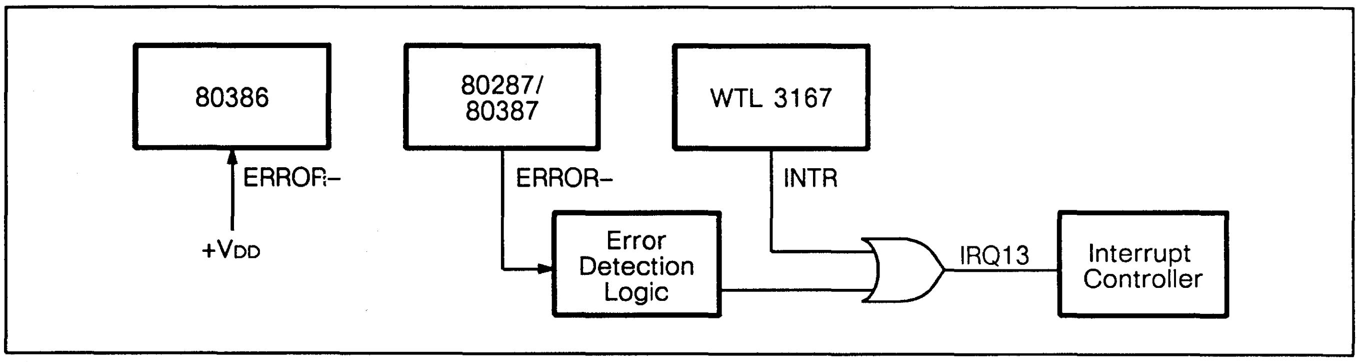


Figure 7. Interrupt output connection in an AT-compatible system

Hardware Designer's Section, continued

SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS

Supply voltage	−0.5 to 7.0 V
Input voltage	−0.5 to VDD
Output voltage	−0.5 to VDD
Storage Temperature Range	−65°C to 150°C
Operating Temperature Range	0°C to 85°C

RECOMMENDED OPERATING CONDITIONS

Parameter		Test Conditions	Commercial		Unit
			Min	Max	
V_{DD}	Supply Voltage		4.75	5.25	V
Tcase	Operating Temperature		0	85	°C

Figure 8.

DC ELECTRICAL CHARACTERISTICS

Parameter		Test Conditions	Commercial		Unit
			Min	Max	
V_{IH}	High-level input voltage	V_{DD} = MAX	2.0		V
V_{IL}	Low-level input voltage	V_{DD} = MIN		0.8	V
V_{IHC}	CLK2 Input high voltage	V_{DD} = MIN/MAX	V_{DD}-.8	V_{DD}+.3	V
V_{ILC}	CLK2 Input low voltage	V_{DD} = MIN		0.8	V
V_{OH}	High-level output voltage	V_{DD} = MIN, I_{OH} = −1.0 mA	2.4		V
V_{OL}	Low-level output voltage	V_{DD} = MIN, I_{OL} = 4.0 mA		0.4	V
I_{IHc}	CLK2 High-level input current	V_{DD} = MAX, V_{IN} = V_{DD}		±10	µA
I_{ILc}	CLK2 Low-level input current	V_{DD} = MAX, V_{IN} = 0V		±10	µA
I_{IH}	High-level input current	V_{DD} = MAX, V_{IN} = V_{DD}		±10	µA
I_{IL}	Low-level input current	V_{DD} = MAX, V_{IN} = 0V		±10	µA
I_{CC}	Supply current	CLK2 = MIN, V_{DD}= MAX		350	mA
WARNING! Remove power before insertion or removal.					

Figure 9. DC electrical characteristics over recommended temperature range

Hardware Designer's Section, continued

AC SWITCHING CHARACTERISTICS

Symbol	Parameter	3167-016		3167-020		3167-025		Unit	Ref Figure	Notes
		Min	Max	Min	Max	Min	Max			
$2T_{CY}$	Clock Cycle Time	62		50		40		ns	12	
T_{CY}	CLK2 Period	31		25		20		ns	12	
T_{CHa}	CLK2 High Time	9		8		7.5		ns	12	At 2V
T_{CHb}	CLK2 High Time	8		7		6.5		ns	12	At 2.5V
T_{CLa}	CLK2 Low Time	9		8		7.5		ns	12	At 2V
T_{CLb}	CLK2 Low Time	7		7		6.5		ns	12	At 0.8V
T_R	Clock Rise Time		8		8		7	ns	12	
T_F	Clock Fall Time		8		8		7	ns	12	
T_1	ADS-, W/R- Setup Time	27		17		15		ns	13,14,15, 16,17	Notes 1, 3, 5
T_2	ADS- Hold Time	4		4		4		ns	13,14,15, 16,17	Notes 1, 3
T_3	$A_{15..2}$, $BE_{2..0}$- Setup Time	22		18		15		ns	13,14,15, 16,17	Notes 1, 3, 5
T_4	$A_{15..2}$, $BE_{2..0}$- W/R- Hold Time	4		4		4		ns	13,14,15, 16,17	Notes 1, 3, 6
T_5	M/IO-, $A_{31..25}$ Setup Time	11		8		7		ns	13,14,15, 16,17	Notes 1, 3, 5
T_6	M/IO-, $A_{31..25}$ Hold Time	4		4		4		ns	13,14,15, 16,17	Notes 1, 3, 6
T_7	$D_{31..0}$ Setup Time	20		20		20		ns	13,14,16	Notes 1, 3, 4, 5
T_8	$D_{31..0}$ Hold Time	2		2		2		ns	13,14,16	Notes 1, 3, 4, 6
T_9	READY- Setup Time	20		12		9		ns	13,14,15, 16,17	Notes 1, 3
T_{10}	READY- Hold Time	4		4		4		ns	13,14,15, 16,17	Notes 1, 3
T_{11}	$D_{31..0}$ Output Delay		48		38		30	ns	15,17	Notes 1, 3, 12
T_{12}	$D_{31..0}$ Valid Output	6		6		5		ns	15,17	Notes 1, 3
T_{13}	$D_{31..0}$ Float Delay		35		27		22	ns	15,17	Notes 3, 8, 13
T_{14}	RESET Setup Time	12		12		10		ns	18	Notes 1, 3
T_{15}	RESET Hold Time	4		4		3		ns	18	Notes 1, 3

Continued next page

Functional Operating Range: V_{DD} = 5V ±5%; T_{case} = 0°C to 85°C

1. All parameters are specified at 1.5V unless otherwise noted
2. All output delays are specified at 1.5V with 50 pf loading unless otherwise noted
3. Relative to CLK2 rising edge
4. Write bus cycle only
5. Referenced to end of cycle when ADS- is de-asserted
6. Hold time reference to end of cycle when ADS is asserted
7. Delay is measured with respect to M/IO-, $A_{31..25}$
8. One CLK2 period
9. Spec only applies when TCB- is high
10. Spec only applies when TCB- is strapped low
11. 85 pf loading
12. 120 pf loading
13. Tri-State timing is guaranteed, but not tested

Figure 10. AC characteristics

Hardware Designer's Section, continued

Symbol	Parameter	3167-016		3167-020		3167-025		Unit	Ref Figure	Notes
		Min	Max	Min	Max	Min	Max			
T_{16}	INTR Output Delay		72		62		50	ns	19	Notes 1, 3
T_{17}	INTR Valid Output	6		6		5		ns	19	Notes 1, 3
T_{18}	MCS- Output Delay		25		20		17	ns	13,17	Notes 1, 7
T_{19A}	RDY- Output Delay (high to low)		26		24		22	ns	13,14,15, 16,17	Notes 1, 3, 9, 11
T_{19B}	RDY- Output Delay (low to high)		26		24		22	ns	14,16,15	Notes 1, 3, 9, 11
T_{20}	RDY- Valid Output	4		4		4		ns	13,15, 16,17	Notes 1, 3, 9, 11
T_{21}	RDY- Tri-State Enable		25		25		20	ns	14,15, 16,17	Notes 3, 10, 11, 13
T_{22}	RDY- Tri-State Disable	5	25	5	25		20	ns	14,15, 16,17	Notes 3, 10, 11, 13
T_{23}	AF32- Tri-State Enable		31		25		20	ns	14,15, 16,17	Notes 3, 10, 13
T_{24}	AF32- Tri-State Disable		31		25		20	ns	14,15, 16,17	Notes 3, 10, 13
T_{25}	AF32- Output Delay		31		25		20	ns	14,15, 16,17	Notes 3, 10
T_{26}	AF32- Valid Output	3		3		2		ns	15	Notes 3, 10

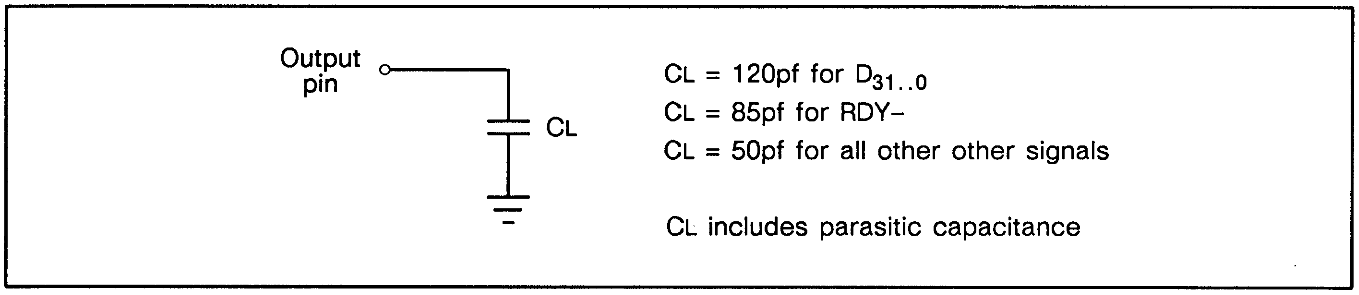
Functional Operating Range: V_{DD} = 5V ±5%; T_{case} = 0°C to 85°C

1. All parameters are specified at 1.5V unless otherwise noted
2. All output delays are specified at 1.5V with 50 pf loading unless otherwise noted
3. Relative to CLK2 rising edge
4. Write bus cycle only
5. Referenced to end of cycle when ADS- is de-asserted
6. Hold time reference to end of cycle when ADS is asserted
7. Delay is measured with respect to M/IO-, $A_{31..25}$
8. One CLK2 period
9. Spec only applies when TCB- is high
10. Spec only applies when TCB- is strapped low
11. 85 pf loading
12. 120 pf loading
13. Tri-State timing is guaranteed, but not tested

Figure 10. AC characteristics, continued

Hardware Designer's Section, continued

TIMING



CL = 120pf for $D_{31..0}$

CL = 85pf for RDY−

CL = 50pf for all other other signals

CL includes parasitic capacitance

Figure 11. Test load for delay measurement

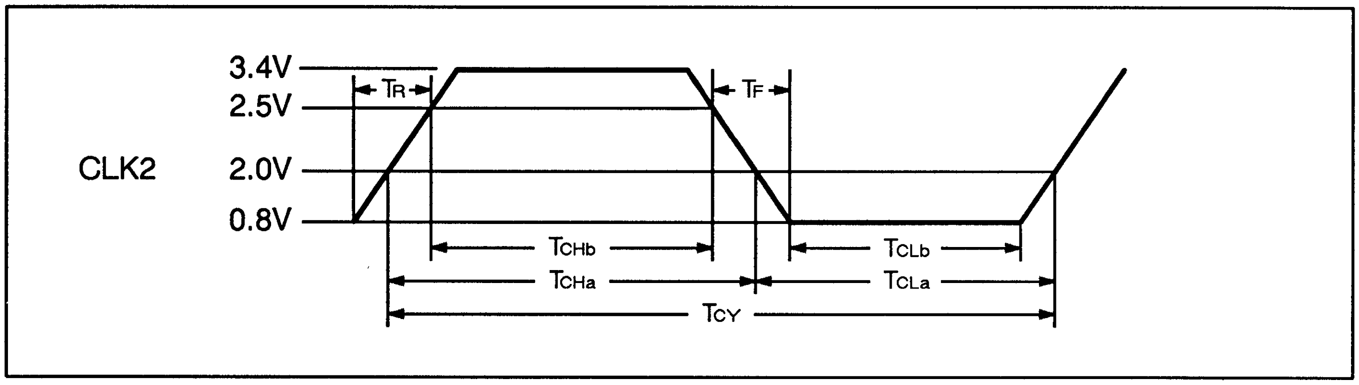


Figure 12. CLK2 timing diagram

Hardware Designer's Section, continued

NON-PIPELINED BUS CYCLES

Figure 13 shows two WTL 3167 write cycles with TCB– high, or NC. Write cycles are performed every time the 80386 broadcasts instructions to the coprocessor. The RDY– output of the WTL 3167 handles the handshaking between the WTL 3167 and the 80386. To acknowledge the current bus cycle, the WTL 3167 asserts RDY– and the 80386 terminates the bus cycle. The first bus write operation does not have the READY– input delayed while the second does. In the delayed READY– write operation, even though the bus does not advance and $D_{31..0}$ is held constant, it is latched in the same cycle it would be if READY– were not delayed. Thus, if the data changes in the time slots indicated in figure 13 with crosshatching, the new data is not used by the WTL 3167.

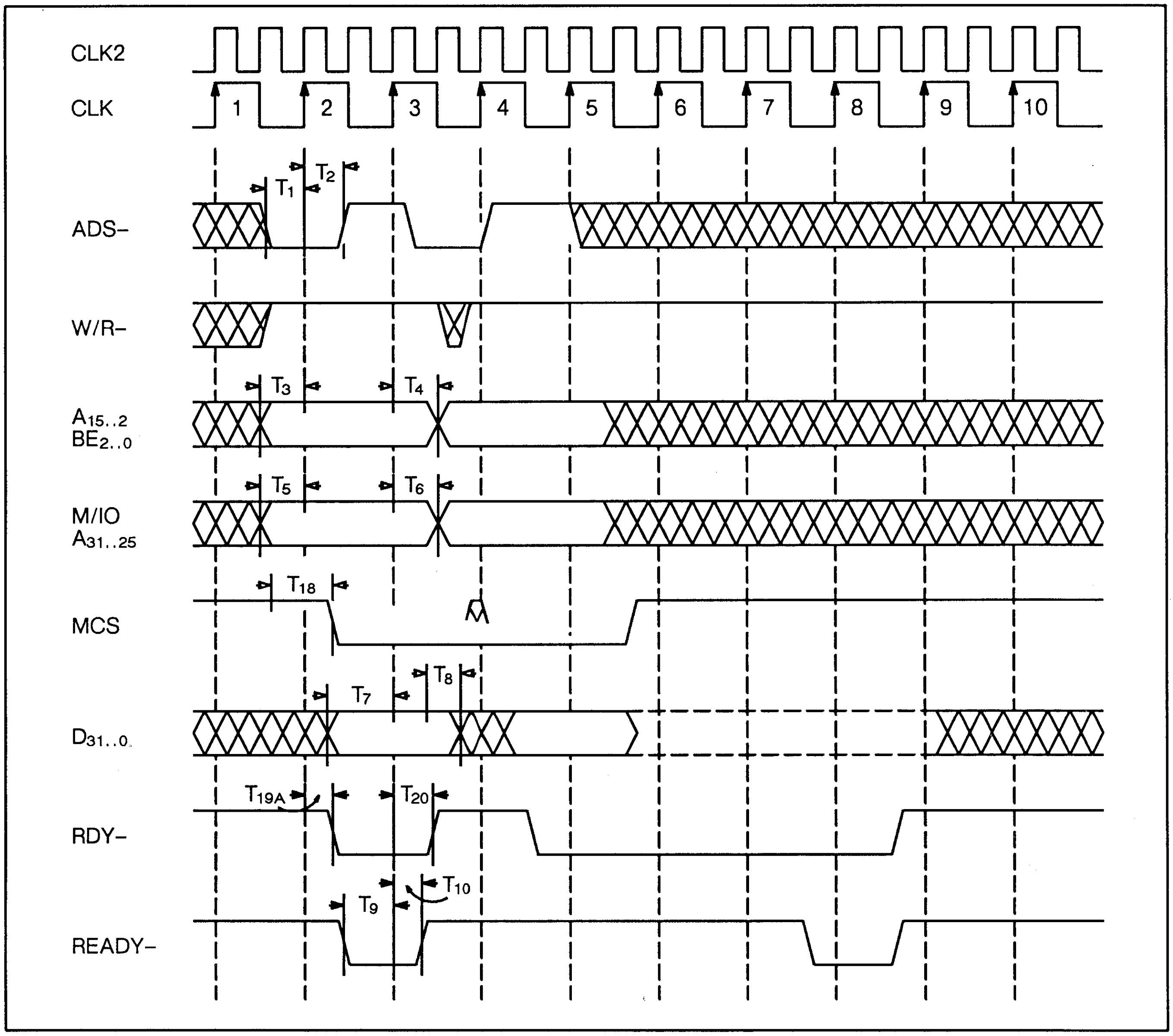


Figure 13. Non-pipelined bus write cycle with and without delayed ready, TCB– high, or NC

Figure 14 shows a new pipelined bus write cycle with TCB- low. The bus cycle now takes a minimum of three clock cycles.

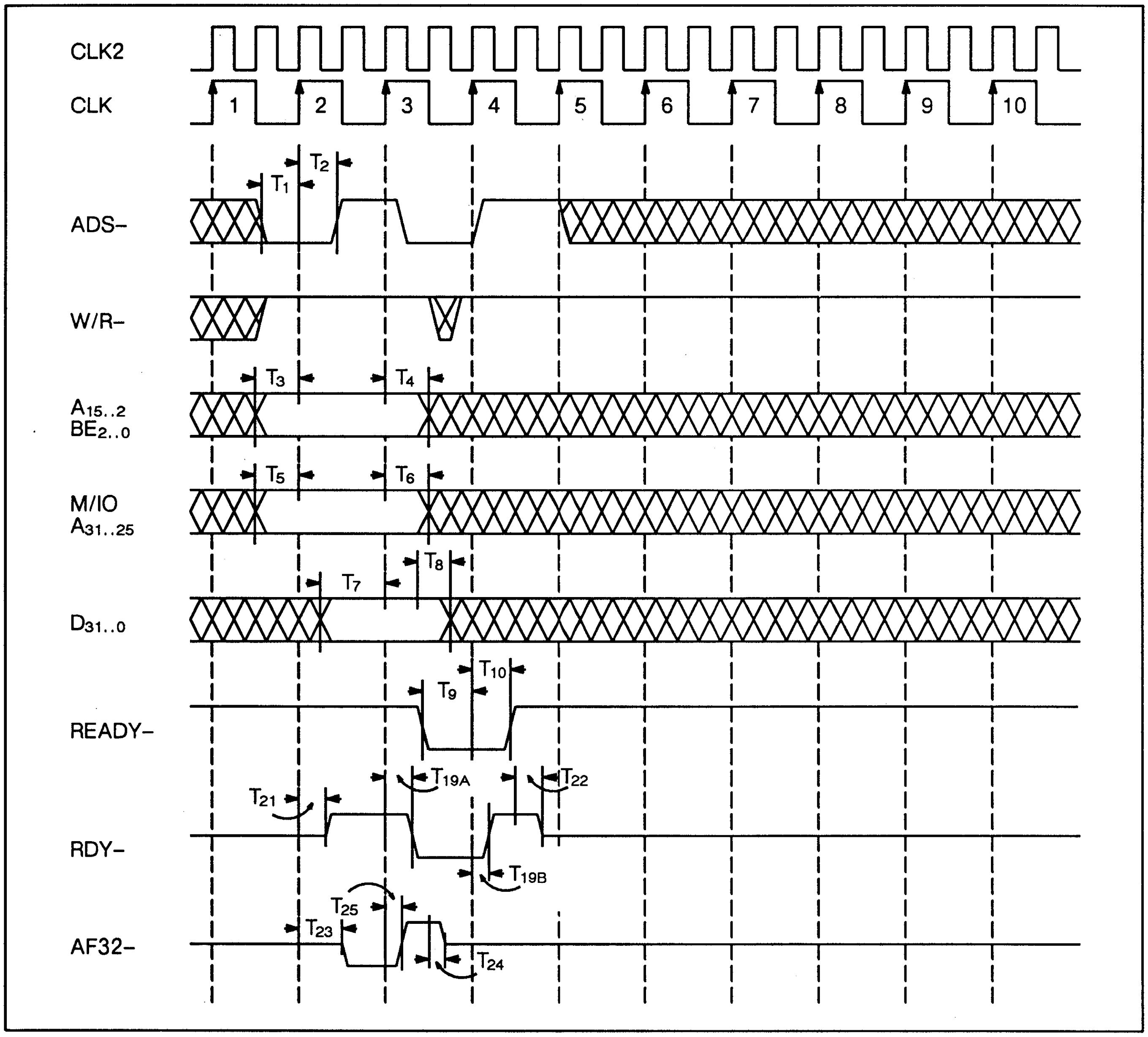


Figure 14. Non-pipelined bus write cycle without delayed ready, TCB- low

Hardware Designer's Section, continued

Read cycles are performed every time data must be read from the WTL 3167 into the 80386. Figure 15 shows a typical WTL 3167 read cycle. At least one wait state is always inserted during a read cycle to allow the WTL 3167 time to respond. As shown in figure 15, the minimum read cycle takes the same number of clock cycles, independent of the value of TCB–.

Wait states are fully transparent to the programmer. If READY– is delayed, the data will continue to be driven until READY– is asserted. When the WTL 3167 receives a bus read operation, it turns on its bus drivers even before the data is ready. The dotted lines in figure 15 shows the time slots during which the WTL 3167 is driving the bus with invalid data. Valid data is only present when RDY– is asserted.

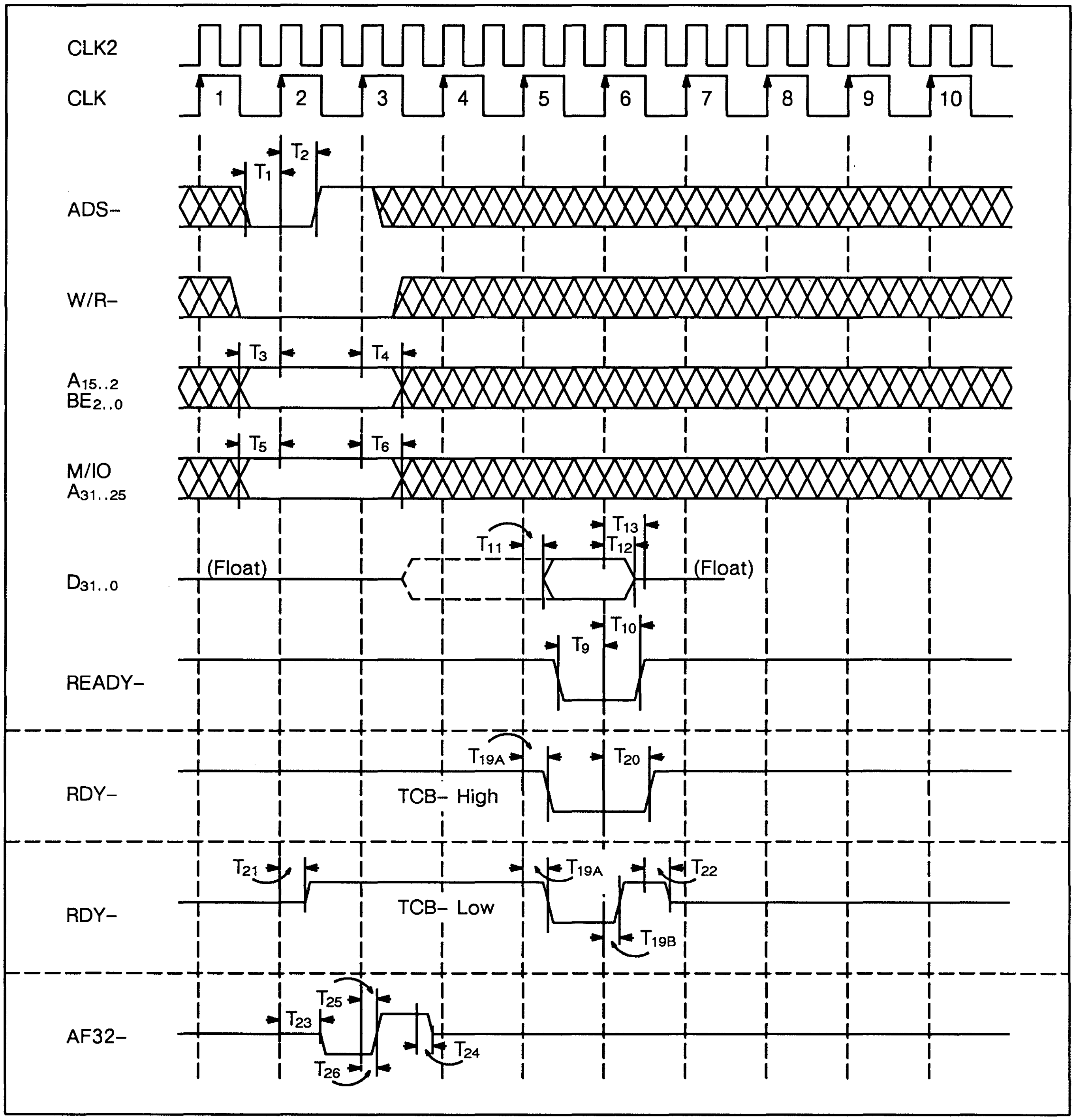


Figure 15. Non-pipelined bus read cycle without delayed ready, with both TCB– high and TCB– low

Hardware Designer's Section, continued

PIPELINED BUS CYCLES

The WTL 3167 is capable of operating with or without address pipelining on a cycle-by-cycle basis. It uses ADS- and READY- from the 80386 to determine when a bus cycle begins. Figure 16 shows a typical pipelined write cycle. The minimum write bus cycle takes the same number of clock cycles, independent of the value of TCB-.

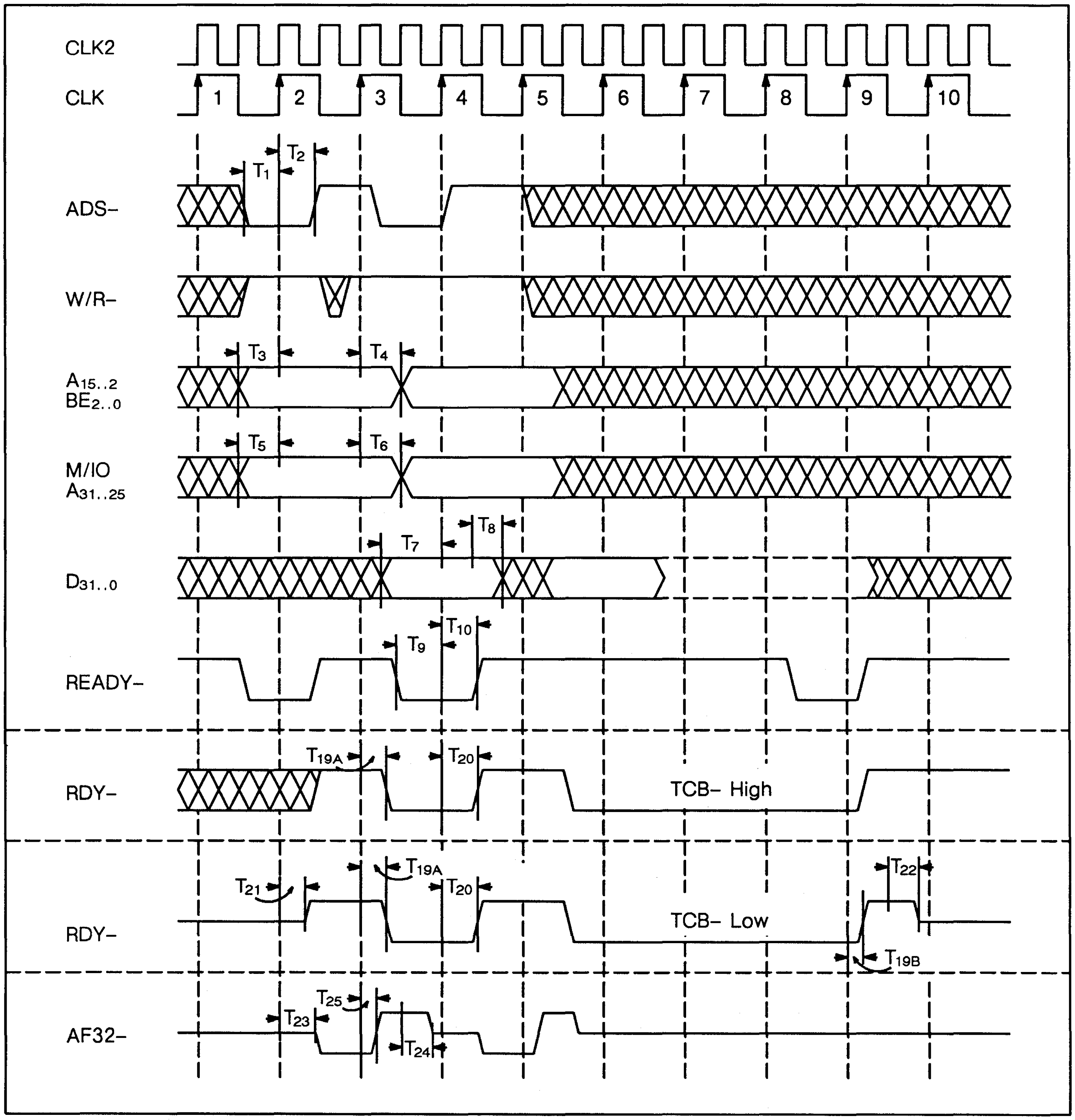


Figure 16. Pipelined bus write cycle without and with delayed ready, and with both TCB- high and TCB- low

Figure 17 shows a typical pipelined read cycle.

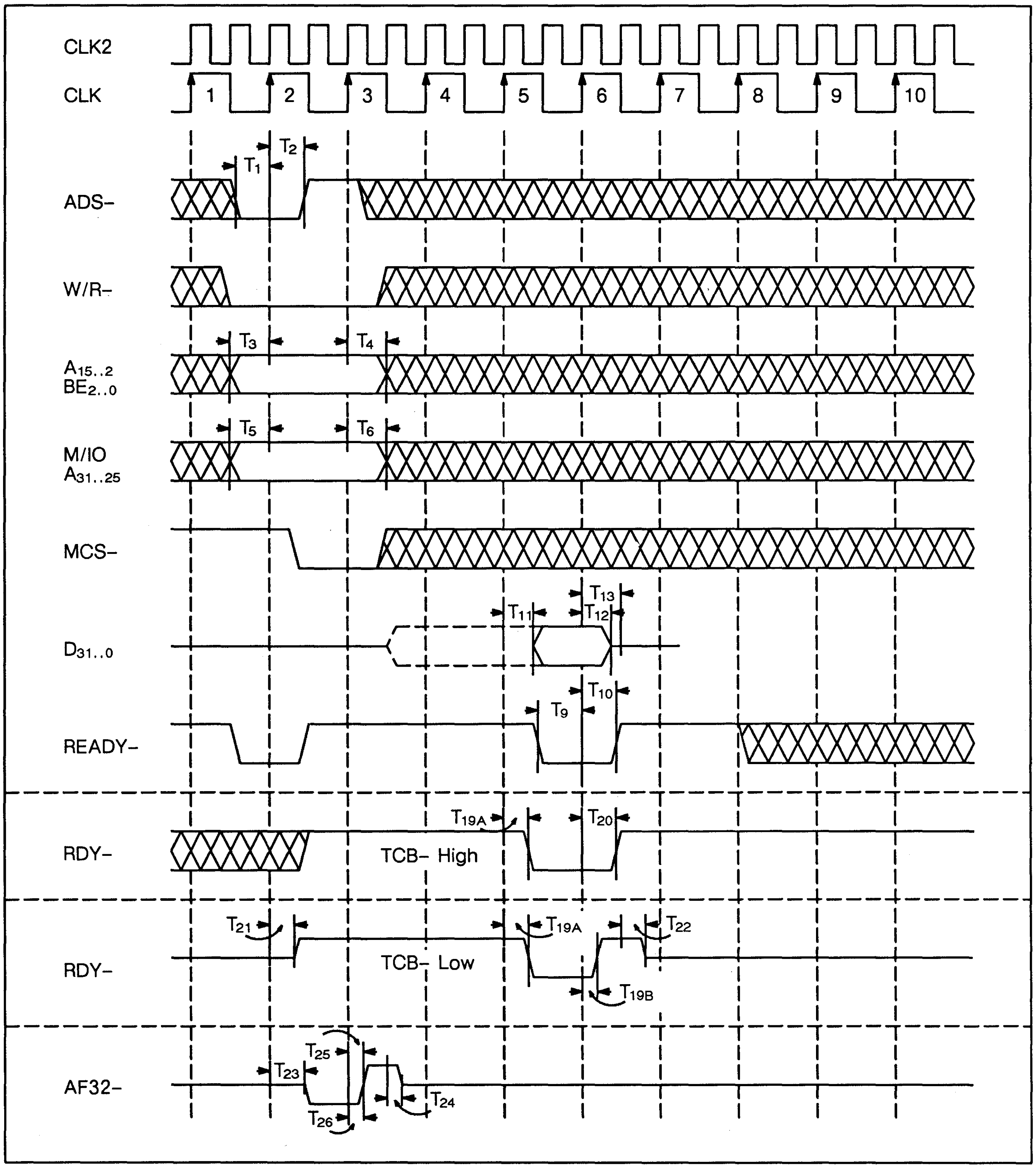


Figure 17. Pipelined bus read cycle without delayed ready, with both TCB– high and TCB– low

Hardware Designer's Section, continued

RESET AND INTERRUPT TIMING

RESET set-up and hold time and interrupt valid delays are shown in figures 18 and 19. RESET must be synchronous with the 80386's clock to guarantee proper operation.

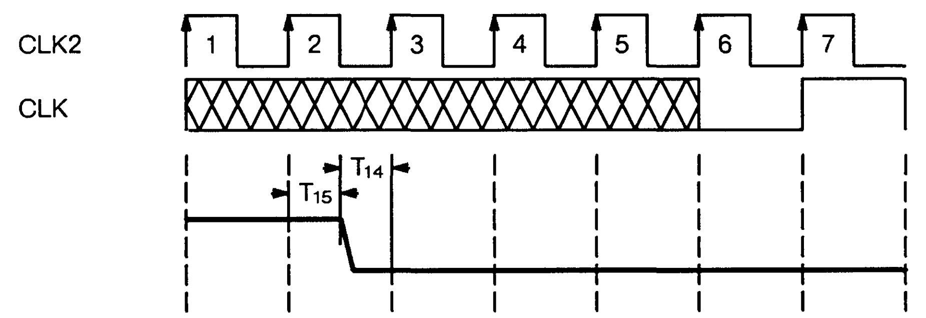


Figure 18. RESET timing

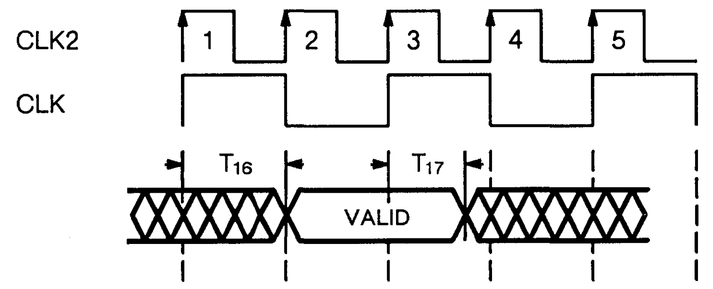


Figure 19. Interrupt timing

Software Tools Overview

The WTL 3167 coprocessor is supported by the UNIX operating system (System V release 3.0). Operating system support includes coprocessor addressing, presence detection at power-up, context-switch handling and emulation. For UNIX operating systems information contact your UNIX supplier. XENIX 386 support is forthcoming.

The WTL 3167 is also supported by Phar Lap, IGC, and AI architects MS-DOS protected mode environments. MS-DOS protected mode environment support for the WTL 3167 includes coprocessor addressing and presence detection.

The WEITEK coprocessor can be supported under real mode MS-DOS as well. OEMs that intend to provide MS-DOS real mode support for the WEITEK coprocessor must refer to the Systems Programmer's section.

C, FORTRAN and Pascal Compilers for the 80386 and WTL 3167 under UNIX V.3 and MS-DOS protected mode are provided by Green Hills, Metaware, Microway, and Silicon Valley Software. Lahey Computer Systems offers an MS-DOS real mode FORTRAN compiler. Metaware also provides MS-DOS real mode C and Pascal compilers. Contact vendors for details.

The WEITEK Coprocessor is fully transparent to the programmer using these compilers, as the floating-point operations are specified with familiar high-level language commands. The compilers include a run-time library for transcendental operations.

Compiler designers and programmers who intend to write WTL 3167 assembly code should refer to the Applications Programmer's section. Systems programmers who need to modify existing operating systems to support the WTL 3167 should refer to the Systems Programmer's section.

TRANSCENDENTAL ROUTINES LIBRARY

WEITEK provides a library of transcendental routines to compiler developers. Routines are available through a simple license agreement.

Vendor	Product	Phone
AI Architects	OS 386 (MS-DOS protected mode environment)	(617) 577-8052
Green Hills Software	C, F, P Compilers (UNIX and MS-DOS protected mode)	(818) 246-5555
IGC	X-AM (MS-DOS protected mode environment)	(408) 986-8373
Lahey Computer Systems	F Compiler (MS-DOS real mode)	(702) 831-2500
Metaware	C, P Compilers (UNIX, MS-DOS real and protected mode)	(408) 429-6382
Microway	C, F, P Compilers (UNIX and MS-DOS protected mode)	(617) 746-7341
Phar-Lap Software	RUN386 (MS-DOS protected mode environment)	(617) 661-1510
Silicon Valley Software	C, F, P Compilers (UNIX and MS-DOS protected mode)	(408) 725-8890
Note: F = Fortran, P = Pascal		

Figure 20. Software tools information

Product	Part Number
UNIX Diagnostics	4800-1167-02
DOS Diagnostics and Macros	4800-1167-03
DOS Demos	4800-1167-04

Figure 21. WEITEK-supplied support software

Applications Programmer's Section

This section provides the information necessary to program the WTL 3167 coprocessor in 80386 assembly language. It is complemented by the *WTL 1167 Software Designer's Guide*. The WTL 3167 is a code-compatible upgrade of the WTL 1167 coprocessor daughter board.

The WTL 3167 internal registers and instruction set are first described in detail. Programming the WEITEK coprocessor can be greatly simplified by defining a set of macro instructions. Macro examples and a simple programming example are part of this section.

REGISTERS

The WTL 3167 provides a register set of 32 single-precision registers, named ws0 through ws31. Pairs of WTL 3167 registers can be used for double-precision operations, allowing up to 16 double-precision registers, numbered wd0, wd2, wd4, ..., wd30. The MSW is stored in the even register and the LSW is stored in the next contiguous odd register (that is, MSW in wsN, LSW in wsN+1). In addition, any 80386 doubleword register can be used to move data, or as the source operand to an arithmetic instruction. The use of register ws0 is restricted. (Refer to page 34 for more details on register ws0.)

PROCESS CONTEXT REGISTER

The WTL 3167 also provides a 32-bit process context register (PCR), which can be written to control rounding modes and exception handling. The context register can also be read to save control settings and read various status flags. The format of this register is defined in figure 23.

MODE SELECTION FIELD

The uppermost byte of the process context register includes the mode selection field and the mode field.

The mode selection field (MDSEL) is used during system initialization to set the mode register in the floating-point chips. If MDSEL is set to 1100 when loading the Context Register, only the EM, CC, and AE fields are updated. If MDSEL is set to 0000 the EM, CC, AE fields, and the rounding mode field (MD) are updated. (See figure 24.)

wd0 →	ws0 (Restricted)	ws1
wd2 →	ws2	ws3
wd4 →	ws4	ws5
⋮	⋮	⋮
wd30 →	ws30	ws31

Figure 22. WTL 3167 register file

31 28	27 24	23 16	15 8	7 0
MDSEL	MD	EM	CC	AE

Figure 23. Process context register

MODE FIELD

The mode field (MD) is used to specify rounding options. Two bits specify one of four rounding modes as defined by the IEEE standard (RN, RZ, RP, RM). A third bit determines the rounding mode used in floating-point-to-integer conversion instructions. It selects either the current rounding mode or the round-to-zero mode. The least-significant bit of the mode field specifies fast mode, (see *IEEE Considerations*, page 42) and must always be set to 1.

31	30	29	28	27	26	25	24
0	0	0	0	RND		IRND	1

RND:	0 0	= Round toward Nearest Value (RN)
	0 1	= Round toward Zero (RZ)
	1 0	= Round toward Positive Infinity (RP)
	1 1	= Round toward Negative Infinity (RM)
IRND:	0	= Integer Rounding based on RND
	1	= Integer Rounding always toward Zero

Figure 24. Mode field

ACCUMULATED EXCEPTION FIELD

The accumulated exception field (AE) contains the five exception flags required by the IEEE standard and other WTL 3167-specific exceptions. The AE field is cleared by writing zeros into the corresponding bits of byte zero of the PCR. The accumulated exception flags are formed by the logic "OR" of the AE field and the current instruction's exception status. The flags accumulate all exceptions which have occurred since the user last cleared the AE field. Exceptions are accumulated regardless of the value of the corresponding exception mask field.

The AE field is shown in figure 25.

7	6	5	4	3	2	1	0
DE	UOE	PE	UE	OE	ZE	EE	IE

DE:	Data Chain Exception
UOE:	Undefined Opcode Exception
PE:	Precision Exception
UE:	Underflow Exception
OE:	Overflow Exception
ZE:	Zero Divide Exception
EE:	Enabled Exception (contains the value of INTR)
IE:	Invalid Operation Exception

Figure 25. Accumulated exception field

EXCEPTION MASK FIELD

The next lower PCR byte is the exception mask field (EM). Seven bits are used to enable exception traps. At the conclusion of an instruction, the accumulated exception field is updated and, if an exception occurred and the corresponding bit in the EM field is set low, the WTL 3167 generates an 80386 interrupt by driving the interrupt request output high. The exception mask byte is shown in figure 26.

Most of the exceptions have the same name as a corresponding 80387 exception and work the same way. The WTL 3167 has an undefined opcode exception, flagged whenever the instruction broadcast by the 80386 is not recognized as a WEITEK instruction. The invalid operation exception is flagged when an invalid operation occurs. The data chain exception is never flagged by the WTL 3167. It has been documented for consistency with the WTL 1167 product. (For a detailed description of the WTL 3167 exception handling refer to *IEEE Considerations*, page 42.)

23	22	21	20	19	18	17	16
DM	UOM	PM	UM	OM	ZM	1	IM

DM:	Data Chain exception Mask
UOM:	Undefined Opcode exception Mask
PM:	Precision exception Mask
UM:	Underflow exception Mask
OM:	Overflow exception Mask
ZM:	Zero Divide exception Mask
IM:	Invalid Operation exception Mask

Figure 26. Exception mask field

Applications Programmer's Section, continued

CONDITION CODE FIELD

The Condition Code Field (CC) is updated only when test or compare instructions are executed. The CC field is updated to reflect the status of the compare operation. At the end of the compare operation the coprocessor status output is encoded and stored in $PCR_{15..8}$. The encoding is shown in figure 27.

INSTRUCTION SET

WTL 3167 instructions can be divided into:

1. Data movement instructions
2. Format conversion instructions
3. Arithmetic instructions
4. Compare and test instructions
5. Sign manipulation instructions

Most WTL 3167 instructions operate on either two WTL 3167 registers or on one WTL 3167 register and the contents of the 80386 data bus. WEITEK coprocessor macro instructions have the format:

OPCODE Source2/Destination, Source1

Source1 and Source2/Destination specify the operand addresses. The operation result is always stored in the same location as Source2. While Source2/Destination always specifies one of the thirty-two WTL 3167 internal registers, Source1 can either specify an internal register (for register-to-register operations), an immediate constant or the content of a 80386 register (for memory-to-register operations).

DATA MOVEMENT INSTRUCTIONS

Data movement instructions move data between the 80386 and a WTL 3167 register, or between two WTL 3167 registers.

Z	C2	C0	Meaning
1	0	0	Equal
0	0	1	Less than
0	0	0	Greater than
1	1	1	Unordered

15	14	13	12	11	10	9	8
X	Z	0	X	X	C2	X	C0

X: not defined
0: bit 13 must be set to zero

Figure 27. Condition code field

```
WFLD     ws1, ws2          ; load ws1 from ws2
WFLD     ws21, EAX         ; load ws21 from EAX
WFLD     ws4, PI           ; load ws4 with constant PI (declared elsewhere)
WFLD     wd4, wd12         ; load ws4 from ws12, then load ws5 from ws13
WFLDCTX  EAX               ; load Context Register from EAX
WFPOP    ws1               ; pop a number from 386 stack to ws1
WFLDSD   ws1, ARRAY, 31    ; load 31 numbers from ARRAY to registers ws1 through ws31
WFLDSD   ws10, ESI, ECX    ; load ECX numbers from ESI to registers starting with ws10
WFST     EDX, ws21         ; store ws21 to EDX
WFSTCTX  EAX               ; store Context register to EAX
WFPUSH   ws1               ; push ws1 onto the 386 stack
WFSTSD   ws0, ARRAY, 32    ; store all 32 registers to ARRAY
WFSTSD   ws10, EDI, ECX    ; store ECX registers from ws10 to EDI
WFSTRL   EAX               ; store revision level to EAX
```

Figure 28. Examples of data movement instructions

FORMAT CONVERSION INSTRUCTIONS

The WTL 3167 provides instructions for converting from any data type supported (single-precision, double-precision, 32-bit integer) to any other type. See figure 29.

```
WFLOAT   ws1, ws10       ; convert integer ws10 to single-precision ws1
WFLOAT   wd4, ws13       ; convert integer ws13 to double-precision wd4
WFLOAT   ws3, EAX        ; convert integer EAX to single-precision ws3
WFLOAT   wd6, EBX        ; convert integer EBX to double-precision wd6
WFLOAT   wd10, 123456    ; load wd10 with the constant 123456.0
WFIX     ws1, ws4        ; convert single-precision ws4 to integer ws1
WFIX     ws3, wd10       ; convert double-precision wd10 to integer ws3
WFIX     ws5, EBX        ; convert single-precision EBX to integer ws5
WFCVT    ws1, wd14       ; convert double-precision wd14 to single-precision ws1
WFCVT    ws8, EBX        ; convert double-precision (EBX, ws1) to single-precision ws8
WFCVT    wd10, ws9       ; convert single-precision ws9 to double-precision wd10
WFCVT    wd26, EAX       ; convert single-precision EAX to double-precision wd26
```

Figure 29. Examples of format conversion instructions

Applications Programmer's Section, continued

ARITHMETIC INSTRUCTIONS

The WTL 3167 provides the four basic arithmetic functions as well as square root. In the subtraction instruction, the Source2/Destination operand is subtracted from the Source1 operand. The reverse subtraction reverses the operands from the standard subtract instruction. The division instruction divides the Source1 operand by the Source2/Destination. The single-precision multiply/accumulate operation multiplies the operands specified by Source1 and Source2 and adds the result to the contents of register ws2. The double-precision multiply/accumulate operation with single-precision inputs multiplies the single-precision operands specified by Source1 and Source2 and adds the result to the contents of register wd2. The double-precision multiply/accumulate operation with double-precision inputs multiplies the double-precision operands specified by Source1 and Source2 and adds the result to the contents of register wd2.

```
WFADD     ws6, ws13       ; add ws13 into ws6
WFADD     wd14, wd20      ; add wd20 into wd14
WFADD     ws3, EAX        ; add EAX into ws3
WFADD     wd2, EBX        ; add (EBX, ws1) into wd2
WFADD     ws1, 9.0        ; add the constant 9.0 into ws1
WFSUBR    ws5, ws30       ; set ws5 to ws30 - ws5
WFSUBR    wd12, wd14      ; set wd12 to wd14 - wd12
WFSUBR    ws3, EDX        ; set ws3 to EDX - ws3
WFSUB     ws5, ws30;      ; set ws5 to ws5 - ws30
WFSUB     ws8, EDX        ; set ws8 to ws8 - EDX
WFMUL     ws1, ws2        ; multiply ws2 into ws1
WFMULN    wd4, wd6        ; set wd4 to (- wd4 × wd6)
WFAMUL    ws5, EAX        ; set ws5 to the absolute value of ws5 × EAX
WFMUL     ws23, 2.0       ; multiply the constant 2.0 into ws23
WFMAC     ws10, ws11      ; add ws10 × ws11 into ws2
WFMAC     ws9, EAX        ; add ws9 × EAX into ws2
WFMACD    ws13, ws29      ; add ws13 × ws29 into wd2
WFMACD    ws1, EBP        ; add ws1 × EBP into wd2
WFMACD    wd12, wd28      ; add wd12 × wd28 into wd2
WFDIVR    ws3, ws5        ; set ws3 to ws5 ÷ ws3
WFDIVR    wd16, wd18      ; set wd16 to wd18 ÷ wd16
WFDIVR    ws2, EAX        ; set ws2 to EAX ÷ ws2
WFDIVR    ws7, PI         ; set ws2 to PI ÷ ws7
WFSQRT    ws3, ws5        ; set ws3 to SQRT(ws5)
WFSQRT    wd10, wd12      ; set wd10 to SQRT(wd12)
```

Figure 30. Examples of arithmetic instructions

COMPARE AND TEST INSTRUCTIONS

Compare and test instructions either compare two floating-point values or compare a single floating-point value to zero. The compare instructions compare Source1 to Source2. Besides comparing the operand to zero, as does the test operation (wftst), test with trap (wftstt) generates an invalid operation exception if the operand is not a valid number (Not a Number, NaN). Test instructions always operate on Source1. Compare and test instructions affect the condition code field of the process context register as shown in figure 27 on page 23.

SIGN MANIPULATION INSTRUCTIONS

The WTL 3167 has two functions that manipulate the sign of a floating-point number: negate and absolute value.

```
WFCMPR   ws3, ws4      ; perform reversed comparison
WFCMPRT  wd8, wd10     ; perform reversed comparison and generate
                       ; invalid exception if one (or both) of the operands is not a
                       ; valid number
WFTST    wd4           ; perform the test of wd4
WFTSTT   ws1           ; perform the test of ws1 and generate
                       ; invalid exception if the operand is not a valid number
```

Figure 31. Examples of compare and test instructions

```
WFNEG    ws1, ws1      ; negate ws1
WFNEG    ws1, ws2      ; set ws1 to -ws2
WFNEG    wd4, wd6      ; set wd4 to -wd6
WFNEG    ws3, EAX      ; set ws3 to -EAX
WFABS    ws3, ws4      ; set ws3 to the absolute value of ws4
WFABS    wd10, wd10    ; coerce wd10 to its absolute value
```

Figure 32. Examples of sign manipulation instructions

Applications Programmer's Section, continued

PROGRAMMING EXAMPLE

The following example shows the code for a 4×4 matrix transformation written using the macros provided by WEITEK.

The matrix coefficients a_{11}, ..., a_{44} are assumed to be already stored in the WTL 3167 registers ws16–ws31.

The variables *x*, *y*, *z*, and *w* are in memory locations *X*, *Y*, *Z*, and *W*. The variables *x*′, *y*′, *z*′, and *w*′ are stored back in memory location *X*, *Y*, *Z*, and *W*.

$$\begin{bmatrix} x & y & z & w \end{bmatrix} \begin{bmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \end{bmatrix} = \begin{bmatrix} x' & y' & z' & w' \end{bmatrix}$$

Figure 33. Matrix multiplication

```
MOV      EAX, X          ; load x into 386 EAX register
WFLD     ws4, EAX        ; load x into ws4
MOV      EAX, Y          ; load y into 386 EAX register
WFLD     ws5, EAX        ; load y into ws5
MOV      EAX, Z          ; load z into 386 EAX register
WFLD     ws6, EAX        ; load z into ws6
MOV      EAX, W          ; load w into 386 EAX register
WFLD     ws7, EAX        ; load w into ws7
WFLD     ws2, ws16       ; move a11 into ws2
WFMUL    ws2, ws4        ; ws2 = a11 × x
WFMAC    ws17, ws5       ; ws2 = (a11 × x) + (a21 × y)
WFMAC    ws18, ws6       ; ws2 = (a11 × x) + (a21 × y) + (a31 × z)
WFMAC    ws19, ws7       ; ws2 = (a11 × x) + (a21 × y) + (a31 × z) + (a41 × w)
WFST     EAX, ws2        ; store x'
MOV      X, EAX          ; store x' into memory location X
WFLD     ws2, ws20       ; move a12 into R2
WFMUL    ws2, ws4        ; ws2 = a12 × x
WFMAC    ws21, ws5       ; ws2 = (a12 × x) + (a22 × y)
WFMAC    ws22, ws6       ; ws2 = (a12 × x) + (a22 × y) + (a32 × z)
WFMAC    ws23, ws7       ; ws2 = (a12 × x) + (a22 × y) + (a32 × z) + (a42 × w)
WFST     EAX, ws2        ; store y'
MOV      Y, EAX          ; store y' into memory location Y
WFLD     ws2, ws24       ; move a13 into ws2
WFMUL    ws2, ws4        ; ws2 = a13 × x
WFMAC    ws25, ws5       ; ws2 = (a13 × x) + (a23 × y)
WFMAC    ws26, ws6       ; ws2 = (a13 × x) + (a23 × y) + (a33 × z)
WFMAC    ws27, ws7       ; ws2 = (a13 × x) + (a23 × y) + (a33 × z) + (a43 × w)
WFST     EAX, ws2        ; store z'
MOV      Z, EAX          ; store z' into memory location Z
WFLD     ws2, ws28       ; move a14 into ws2
WFMUL    ws2, ws4        ; ws2 = a14 × x
WFMAC    ws29, ws5       ; ws2 = (a14 × x) + (a24 × y)
WFMAC    ws30, ws6       ; ws2 = (a14 × x) + (a24 × y) + (a34 × z)
WFMAC    ws31, ws7       ; ws2 = (a14 × x) + (a24 × y) + (a34 × z) + (a44 × w)
WFST     EAX, ws2        ; store w'
MOV      W, EAX          ; store w' into memory location W
```

Figure 34. Matrix transformation in assembly language

Applications Programmer's Section, continued

INSTRUCTION SUMMARY

Figure 35 summarizes the WTL 3167 instruction set macros. All WTL 3167 register names begin with "w". We follow the "w" with either "s" for single, "d" for double, or "x" meaning either "s" or "d". The register name ends with the letter "t" or "f". "t" stands for "to" and "f" stands for "from". For most instructions, wxt is the destination register and wxf is the source register.

```
Data Movement
WFLD      wst, wsf             ; load: wst = wsf
WFLD      wst, data            ; load: wst = 386 data
WFLD      wdt, wdf             ; load: wdt = wdf
WFLDCTX   ereg                 ; load: CTX = 386 E-register
WFPOP     wst                  ; pop wst from the 386 stack
WFPOP     wdt                  ; pop two doublewords from the 386 stack to wdt
WFLDSD    wst, addr, count     ; block move: wst array = 386 memory

WFST      ereg, wst            ; store: 386 E-register = wst
WFST      ereg, wst, opcode    ; store: 386 ereg = ereg <opcode> wst
WFSTCTX   ereg                 ; store: 386 E-register = CTX
WFSTCTX   ereg, opcode         ; store: 386 ereg = ereg <opcode> CTX
WFPUSH    wst                  ; push wst onto the 386 stack
WFPUSH    wdt                  ; push wdt (two doublewords) onto the 386 stack
WFSTSD    wst, addr, count     ; block move: 386 memory = wst array
WFSTRL    EAX                  ; store revision level to EAX

Format Conversion
WFLOAT    wxt, wsf             ; convert integer wsf to floating wxt
WFLOAT    wxt, data            ; convert integer 386 data to floating wxt
WFIX      wst, wxf             ; convert floating wxf to integer wst
WFIX      wst, data            ; convert floating (386 data) to integer wst

WFCVT     wst, wdf             ; convert wdf to wst
WFCVT     wst, data            ; convert double-precision (386 data and ws1) to wst
WFCVT     wdt, wsf             ; convert wsf to wdt
WFCVT     wdt, data            ; convert single-precision 386 data to wdt
```

(continued next page)

Figure 35. The WTL 3167 instruction set macros

Four-Function Arithmetic

```
WFADD    wxt, wxf        ; add: wxt = wxt + wxf
WFADD    wxt, data       ; add: wxt = wxt + (386 data)

WFSUBR   wxt, wxf        ; reversed subtract: wxt = wxf - wxt
WFSUBR   wxt, data       ; reversed subtract: wxt = (386 data) - wxt
WFSUB    wxt, wxf        ; subtract: wxt = wxt - wxf (1)
WFSUB    wxt, data       ; subtract: wxt = wxt - (386 data) (1)

WFMUL    wxt, wxf        ; multiply: wxt = wxt × wxf
WFMUL    wxt, data       ; multiply: wxt = wxt × (386 data)
WFMULN   wxt, wxf        ; negative multiply: wxt = -wxt × wxf
WFMULN   wxt, data       ; negative multiply: wxt = -wxt × (386 data)
WFAMUL   wxt, wxf        ; absolute multiply: wxt = |wxt × wxf|
WFAMUL   wxt, data       ; absolute multiply: wxt = |wxt × (386 data)|

WFMAC    wst, wsf        ; multiply and accumulate: ws2 = ws2 + wst × wsf
WFMAC    wst, data       ; multiply and accumulate: ws2 = ws2 + wst × (386 data)
WFMACD   wst, wsf        ; multiply and accumulate: wd2 = wd2 + wst × wsf (1)
WFMACD   wst, data       ; multiply and accumulate: wd2 = wd2 + wst × (386 data) (1)
WFMACD   wdt, wdf        ; multiply and accumulate: wd2 = wd2 × wdf (1)

WFDIVR   wxt, wxf        ; reversed divide: wxt = wxf ÷ wxt
WFDIVR   wxt, data       ; reversed divide: wxt = (386 data)/wxt
WFSQRT   wxt, wxf        ; square root: wxt = sqrt(wxf) (1)
WFSQRT   wxt, data       ; square root: wxt = sqrt(data) (1)
```

Compare and Test

```
WFCMPR   wxt, wxf;       ; reversed compare: set CTX flags for (wxf - wxt)
WFCMPR   wxt, data       ; reversed compare: set CTX for (386 data) - wxt
WFCMPRT  wxt, wxf        ; reversed compare with trap: set CTX flags for (wxf - wxt)
WFCMPRT  wxt, data       ; reversed compare with trap: set CTX for (386 data) - wxt

WFTST    wxf             ; test: set CTX flags for (wxf - 0)
WFTST    data            ; test: set CTX flags for (386 data) - 0
WFTST    ata, ws1        ; test: set CTX flags for double-precision (386 data, ws1) - 0
WFTSTT   wxf             ; test with trap: set CTX flags for (wxf - 0)
WFTSTT   data            ; test with trap: set CTX flags for (386 data) - 0
WFTSTT   data, ws1       ; test with trap: set CTX flags for (386 data, ws1) - 0
```

Sign Manipulation

```
WFNEG    wxt, wxf        ; negate: wxt = -wxf
WFNEG    wxt, data       ; negate: wxt = -(386 data)
WFABS    wxt, wxf        ; absolute value: wxt = |wxf|
WFABS    wxt, data       ; absolute value: wxt = |386 data|
```

Paging Directives

```
WFSPAGE                  ; force next wfld/wfst to single-precision page
WFDPAGE                  ; force next wfld/wfst to double-precision page
```

(1) These instructions are not available on the WTL 1167

Figure 35. The WTL 3167 instruction set macros, continued

Applications Programmer's Section, continued

INSTRUCTION SET—MACHINE'S POINT OF VIEW

The WTL 3167 is a memory-mapped device. The coprocessor is mapped in the physical memory area ranging from C0000000 hex to C000FFFF hex. A given address in this memory area selects the coprocessor, indicates the instruction which the WTL 3167 has to perform, and specifies the location of Source1 and Source2/Destination. Figure 36 shows how the WTL 3167 views a 32-bit address word.

COPROCESSOR SELECT

The most-significant 16 bits of the physical address identify a coprocessor instruction. If the upper bits do not fall in the C000–C1FF range, the address does not specify a WEITEK command and is then ignored by the WTL 3167. To ensure compatibility with future devices, we recommend that you set the coprocessor select field to C000 when specifying a WTL 3167 instruction.

OPCODE FIELD

The next six bits specify the coprocessor instruction to be executed. Figure 37 provides the binary and hexadecimal offset, the hexadecimal number obtained by placing the six opcode bits into the opcode field of the address, for the WTL 3167 instructions.

OPERAND FIELDS

The five bits of the Source1 and Source2/Destination fields identify the registers that will provide sources and destination for the instruction. If Source1 is set to zero, the Source1 data is moved over the system data bus. In order to take advantage of the 80386 block-move instruction (refer to *Arrangement of Fields to Accommodate Block Moves* on page 34) the Source1 field is split into a three-bit and a two-bit field. The two-bit field occupies the two least-significant bits of the address.

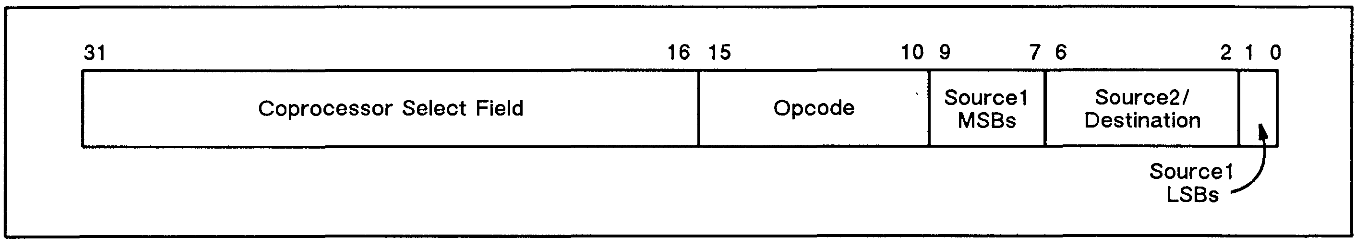


Figure 36. WTL 3167 view of 80386 address word

INSTRUCTION ENCODING

Figures 37 and 38 show the mnemonic and the encoding for both opcode and operands (Source1 and Source2/destination).

Opcode Mnemonic	Binary Value	Hex Offset	Opcode Mnemonic	Binary Value	Hex Offset
ADD.S	000000	0000	ADD.D	100000	8000
LOAD.S	000001	0400	LOAD.D	100001	8400
MUL.S	000010	0800	MUL.D	100010	8800
STOR.S	000011	0C00	STOR.D	100011	8C00
SUBR.S	000100	1000	SUBR.D	100100	9000
DIV.S	000101	1400	DIV.D	100101	9400
MULN.S	000110	1800	MULN.D	100110	9800
FLOAT.S	000111	1C00	FLOAT.D	100111	9C00
CMPT.S	001000	2000	CMPT.D	101000	A000
TSTT.S	001001	2400	TSTT.D	101001	A400
NEG.S	001010	2800	NEG.D	101010	A800
ABS.S	001011	2C00	ABS.D	101011	AC00
CMP.S	001100	3000	CMP.D	101100	B000
TST.S	001101	3400	TST.D	101101	B400
AMUL.S	001110	3800	AMUL.D	101110	B800
FIX.S	001111	3C00	FIX.D	101111	BC00
CVTS.D	010000	4000	LDCTX	110000	C000
CVTD.S	010001	4400	STCTX	110001	C400
MAC.S	010010	4800	MACD.S	110010	C800
SQRT.S	010011	4C00	SQRT.D	110011	CC00
MACD.D	010100	5000	LOADD.D	110100	D000
SUB.S	010101	5400	STORD.D	110100	D000
			SUB.D	110101	D400
Note: .S in the opcode field stands for single-precision while .D stands for double-precision.					

Figure 37. Opcode encoding

Applications Programmer's Section, continued

GENERATING WTL 3167 INSTRUCTIONS WITH 80386 MEMORY MOVES

Suppose that two single-precision numbers, stored in the WTL 3167 registers F1 and T2, need to be added and the result stored in T2. Since the coprocessor is mapped in the memory range C0000000–C000EFFF hex, the instruction will be specified by the following coprocessor select, opcode, and operand address fields:

COPROCESSOR SELECT =	C000 0000 hex
OPCODE = ADD.S =	0000 hex
Source1 = F1 =	01 hex
Source2/Destination = T2 =	08 hex

The 80386 address specifying the floating-point instruction is then given by:

$A_{31}..A_0$ = C0000009 hex

An 80386 move instruction which generates a physical address of C0000009 hex causes the WTL 3167 to execute the floating-point addition.

A LOW-LEVEL REPRESENTATION OF THE WTL 3167 INSTRUCTION SET

We can see from the previous example that the single-precision WFADD instruction appears in the 80386's memory space as an array of 1024 consecutive addresses, one for each combination of 32×32 operands. Thus, there is a natural low-level representation of WTL 3167 instructions as arrays of memory addresses. The array starting location is determined by the specific opcode shown in figure 37. The elements of the array can be represented by the operand offset values provided in figure 38.

Source2/ Destination Mnemonic	Decimal Value	Hex Offset	Source1 Mnemonic	Decimal Value	Hex Offset
T0	0	00	F0	0	00
T1	1	04	F1	1	01
T2	2	08	F2	2	02
T3	3	0C	F3	3	03
T4	4	10	F4	4	80
T5	5	14	F5	5	81
T6	6	18	F6	6	82
T7	7	1C	F7	7	83
T8	8	20	F8	8	100
T9	9	24	F9	9	101
T10	10	28	F10	10	102
T11	11	2C	F11	11	103
T12	12	30	F12	12	180
T13	13	34	F13	13	181
T14	14	38	F14	14	182
T15	15	3C	F15	15	183
T16	16	40	F16	16	200
T17	17	44	F17	17	201
T18	18	48	F18	18	202
T19	19	4C	F19	19	203
T20	20	50	F20	20	280
T21	21	54	F21	21	281
T22	22	58	F22	22	282
T23	23	5C	F23	23	283
T24	24	60	F24	24	300
T25	25	64	F25	25	301
T26	26	68	F26	26	302
T27	27	6C	F27	27	303
T28	28	70	F28	28	380
T29	29	74	F29	29	381
T30	30	78	F30	30	382
T31	31	7C	F31	31	383

Figure 38. Operands encoding

Returning to our example, the single-precision WFADD instruction has the low-level mnemonic ADD.S in figure 37. If we declare ADD.S as a memory array starting at location 0C0000000, and we declare the operand offsets F1 as 01h and T2 as 08h, our coding of the ADD.S ws2, ws1 instruction becomes:

```
MOV ADD.S[T5 + F1], AL
```

It is important to notice that when Source1 is set to zero (F0), it actually specifies an operand not resisting is the register file and being provided by the data bus. When Source2/Destination is set to zero (T0), it looks like any other register in the register file.

AVOIDING OVERLAPPED DOUBLEWORD REFERENCES

There is a pitfall to avoid while using the low-level mnemonics to code WTL 3167 instructions: you will obtain incorrect results if you indiscriminately choose to access WTL 3167 memory with a doubleword transfer when a byte-sized transfer would have sufficed. The pitfall applies to instructions involving only WTL 3167 registers, and not any data on the 80386 bus, as in the previous example.

For example, suppose we want to provide a low-level encoding for WFAMUL ws8,ws14. In this instruction, the 80386 data bus is ignored by the WTL 3167. So any memory access to the address AMUL.S[T8 + F14] will cause the multiplication to be performed. The instruction with the shortest encoding is:

```
MOV AMUL.S[T8 + F14], AL
```

To understand the pitfall, let us see what happens if we instead code an unnecessary doubleword memory access:

```
MOV AMUL.S[T8 + F14], EAX
```

By adding the offsets of AMUL.S, T8, and F14 to the WTL 3167 base address 0C0000000h, we find that we have encoded a doubleword write to memory location 0C00039A2h. The memory address is not a multiple of four, so the doubleword being written is not aligned on a doubleword boundary. When doublewords are not aligned, the 80386 splits the memory write into two operations, as shown in figure 39. First it will write the bottom half of EAX to the top half of 0C00039A0; then it will write the top half of EAX to the bottom half of 0C00039A4. The WTL 3167 will misinterpret this as two consecutive floating-point instructions, instead of the single AMUL.S that was intended.

ARRANGEMENT OF FIELDS TO ACCOMMODATE BLOCK MOVES

The 80386 has an instruction, REP MOVSD, that moves a block of doublewords from one memory location to another. Three 80386 registers must be initialized before the REP MOVSD is executed: ECX holds the number of doublewords to be moved, ESI points to the source of the move, and EDI points to the destination. (The instruction can also be executed with 8086-style addressing, using the registers CX, SI, and DI.)

The operand fields of a WTL 3167 address have been specifically designed so that a WTL 3167 address can be given as either the source or the destination to a REP MOVSD instruction. Due to the positioning of the destination operand slot two bits from the bottom of the address, T-offsets increase by four for successive WTL 3167 registers. The registers appear as successive doubleword addresses in the 80386 memory space, allowing the REP MOVSD instruction to work correctly.

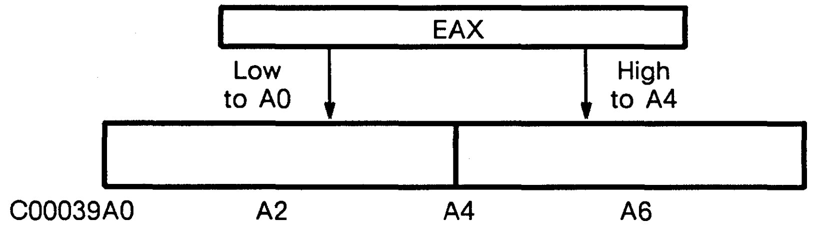


Figure 39. Erroneous overlapped doubleword transfer to the WTL 1167 space

Applications Programmer's Section, continued

SPECIAL INSTRUCTION FORMS NOT PROVIDED BY THE MACRO SET

The low-level interface presented in this chapter allows for some interesting possibilities not offered by the macro set. Be forewarned, however, that most WTL 3167 software emulation packages will not duplicate the functionality of the low-level interface. Thus, if there is the possibility that your program will run in a system that emulates the WTL 3167 in software, you should restrict yourself to the standard forms of the macro set.

Single-precision vector arithmetic is accomplished by applying the 80386 block move instruction REP MOVSD to a WTL 3167 address involving arithmetic instead of loading or storing. For example, the following instruction sequence multiplies each element of the doubleword array VECTOR in 80386 memory, into the corresponding element of the WTL 3167 register array ws11 through ws20:

```
MOV      ECX, 10                 ; load the number of elements of the vector array
MOV      ESI, OFFSET VECTOR      ; point to the memory vector
MOV      EDI, OFFSET MUL.S[T10]  ; point to the WFMUL address for ws10
REP MOVSD                        ; multiply each VECTOR element into a WTL 3167 register
```

Figure 40.

Similarly, the REP STOSD instruction could be used to fill an array of WTL 3167 registers with the same value, or to perform arithmetic of the same value applied to consecutive WTL 3167 registers. For example, the following sequence clears the entire WTL 3167 register set to zero:

```
SUB      EAX, EAX                ; integer 0 is also floating-point 0
MOV      ECX, 32                 ; there are 32 registers to fill
MOV      EDI, OFFSET LOAD.S[T0]  ; first STOSD will load EAX=0 to ws0
REP STOSD                        ; load each WTL 1167 register with a zero value
```

Figure 41.

The following sequence multiplies each of the registers ws11 through ws18 by two:

```
MOV      EAX, 40000000h          ; load single-precision "2.0" into EAX
MOV      ECX, 8                  ; there are 8 registers to multiply
MOV      EDI, OFFSET MUL.S[T11]  ; first STOSD will multiply EAX into ws11
REP STOSD                        ; multiply each of 8 registers by EAX
```

Figure 42.

PHYSICAL VERSUS LOGICAL ADDRESSES

The 80386 has three distinct address spaces: logical, linear, and physical. A logical address consists of a selector and an offset. The segmentation unit translates the logical address space into a 32-bit linear address space. If the paging unit is not enabled, then the 32-bit linear address corresponds to the physical address. Otherwise, the paging unit translates the linear address space into the physical address space. The physical address is what appears on the address pins and is responsible for specifying WTL 3167 instructions. (For more details refer to the Intel 80386 data sheet and the *80386 Programmer's Reference Manual*). The logical to physical address translation is fully transparent to the applications programmer. Applications programmers need only to know which logical addresses will be mapped into WTL 3167 physical addresses.

WTL 3167 MS-DOS REAL MODE ADDRESSING

This paragraph describes how logical addresses are mapped into physical addresses for the WTL 3167 in the MS-DOS environment. OEMs that support the WTL 3167 under real mode MS-DOS must implement the same address translation scheme described in this paragraph.

While the 8086 can form addresses only up to 20 bits long, the 80386 has access to 21 bits in real-address mode. For example, assuming a selector value equal to 0FFFF hex and an offset of 0FFFF hex, in real mode the effective address would be 10FFEF hex (Selector × 6 + Offset = FFFF0 hex + FFFF hex = 10FFEF hex). The 8086 would truncate the high order bit, wrapping this address to 0FFEF hex, while the 80386 would preserve the entire 21 bits. 80386 users then have access to extra 65520 bytes of memory that do not conflict with the traditional one megabyte address range for MS-DOS. Such extra memory is enough to accommodate the WEITEK coprocessor.

In MS-DOS the WTL 3167 resides at logical base address 100000 hex with instructions mapped into addresses 100000 hex to 10EFFF hex. The 80386 paging unit is then used to map logical addresses 100000 hex through 10EFFF hex to the physical address space ranging from C0000000 hex through C000EFFF hex. More details on how to implement this address translation scheme are presented in the Systems Programmer's Section. Thanks to this address translation, real mode programs can access the WTL 3167 coprocessor. MS-DOS applications can address the WTL 3167 by setting a segment register (for example fs) to FFFF hex, adding an address offset of 0010 hex (to access base address 100000 hex), then executing move instructions that generate coprocessor addresses between 100000 hex and 10EFFF hex.

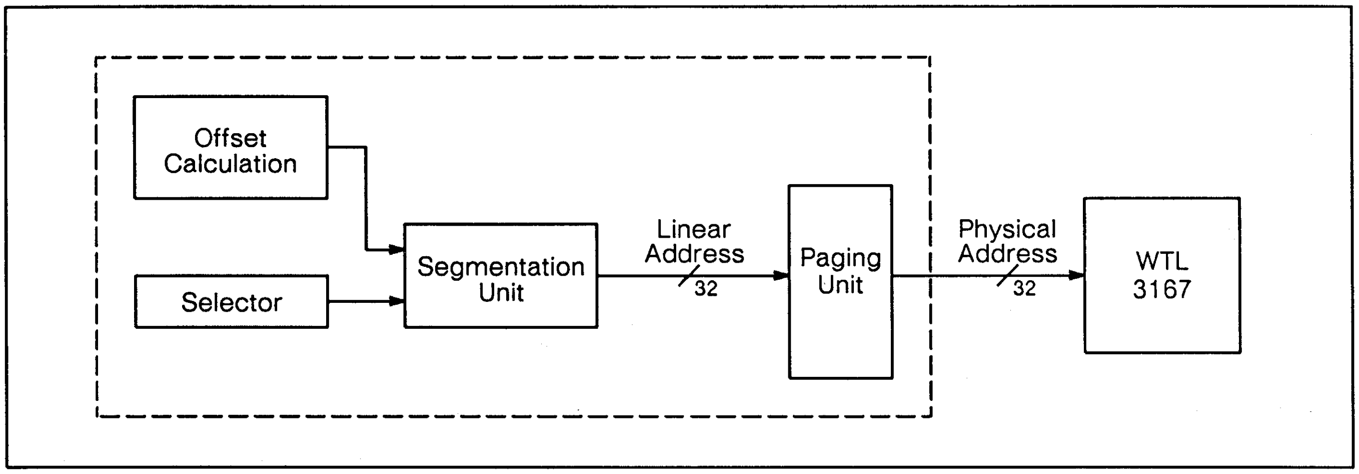


Figure 43. Address translation

Applications Programmer's Section, continued

Assuming that the single-precision add instruction of registers ws1 and ws2 has to be coded under MS-DOS, the following instruction would do:

```
mov FFFF:0019h, al
```

The address FFFF:0019h is derived as follows:

```
COPROCESSOR SEGMENT =           FFFF:0000h
OFFSET TO ADDRESS 100000h =     0010h
OPCODE = ADD.S =                0000h
Source1 = F1 =                  0001h
Source2/Destination = T2 =      0008h

ACTUAL ADDRESS GENERATED =      FFFF:0019h
```

The segmented address, written FFFF:0019h, is equivalent to address 100009 hex. The use of the fs segment and the offset of 10 hex is pre-programmed into the WTL 3167 real mode macro set.

EXECUTION TIMES FOR INDIVIDUAL INSTRUCTIONS

To estimate WTL 3167 performance, the table in figure 44 may be used. The double-precision memory-to-register estimates include a load ws1 instruction.

The figures below assume that new instructions are sent to the WTL 3167 within six cycles of the acknowledgment of a transfer by the coprocessor.

ESTIMATED TIMES FOR TRANSCENDENTAL FUNCTIONS

Figure 45 gives the execution times for procedures in our library of transcendental functions. The exact times may vary according to the values of the operands handed to the functions; the times in the table are average times. Transcendental routines are provided to compiler vendors with WTL 3167 support.

Instruction Type	Single-Precision Register-to-Register	Double-Precision Register-to-Register
LOAD, Compare, ABS	3 cycles	3 cycles
ADD, SUB, NEG, Conversion	6 cycles	6 cycles
MUL	6 cycles	10 cycles
AMUL	9 cycles	13 cycles
MULN	12 cycles	16 cycles
DIV	38 cycles	66 cycles
SQRT	60 cycles	118 cycles
MAC	12 cycles	16 cycles
MACD.S	12 cycles	
STORE*	3 cycles	

*Store operations require a variable number of cycles because they cannot be performed if any other operation is in progress.

Figure 44. Latency

Function	Single-Precision	Double-Precision	Absolute Accuracy(1)	Relative Accuracy(17)	Monotonicity
SQRT (2)	117 cycles	285 cycles	n/a (3)	5 ULPs	TDT (4)
SIN (5)	146	292	1.6 ULPs (18)	5	TDT
COS (5)	140	285	2.2	5	TDT
ATAN (6)	157	398	3.0	5	TDT
EXP (7)	179	401	2.2	5	TDT
LOG (8)	171	365	2.7	5	TDT
TAN (9)	188	340 (10)	n/a (3)	5 ULPs	n/a (11)
COTAN (9)	150	372 (10)	n/a (3)	5 ULPs	n/a (11)
ASIN (12)	175	467	n/a (3)	5 ULPs	n/a (11)
ACOS (12)	175	467	n/a (3)	5 ULPs	n/a (11)
SINH (13)	185	400 (14)	n/a (3)	5 ULPs	n/a (11)
COSH (13)	185	400 (14)	n/a (3)	5 ULPs	n/a (11)
TANH (15)	194	350	n/a (3)	5 ULPs	n/a (11)
REM (6)	n/a	n/a	n/a (3)	5 ULPs	n/a (11)
MOD (6)	n/a	n/a	n/a (3)	5 ULPs	n/a (11)
ASCII→BINARY (16)			.01 ULP	.01 ULP	To .01 ULP
BINARY→ASCII (16)			.01 ULP	.01 ULP	To .01 ULP

Notes:

1. As determined by Alex Liu's "Elefunt" program
2. Square root can be implemented much faster using the SQRT instruction. The routine is used when running code written for the WTL 1167. The number shown is an average for 100,000 uniformly distributed numbers from 0 through 50,000
3. Absolute accuracy tests do not exist for these functions
4. TDT is an abbreviation for "to the degree tested"
5. Average for 50,000 uniformly distributed numbers from 0 through $\pi/4$, 25,000 uniformly distributed numbers from $\pi/4$ through $\pi/2$, and 25,000 uniformly distributed numbers in the range of $\pi/2$ through π
6. Average for 100,000 uniformly distributed numbers from −1 through 1
7. Average for 100,000 uniformly distributed numbers from −10 through 10
8. Average for 100,000 uniformly distributed numbers from e^{-10} through e^{10}.
9. Average from 0 to 4.1×10^3
10. Average from 0 to 6.7×10^7
11. Monotonicity has yet to be determined
12. Average from 0 to 1
13. Average from 0 to 89
14. Average from 0 to 710
15. Average from 0 to ∞
16. See figures 46 and 47
17. As determined by Cody and Waite's transcendental routines.
18. ULP is an abbreviation for "units in the last place"

Figure 45. Average execution times for transcendental functions

Applications Programmer's Section, continued

String	Single	Double	
0	310	310	
1	384	416	
1.23456	704	704	6 Digits
123456.	672	672	
123456789012345.	1120	1152	15 Digits
1234567890.12345	1088	1184	
12345678901234567890.	1376	1696	20 Digits (and optional exponent)
1234567890.1234567890	1344	1696	
1234567890.123456789e10	1568	1888	
12345678901234567890.e10	1568	1856	
1.2345678901234567890e38	1600	1856	
12.345678901234567890e-38	1664	2048	
1.23456e15	832	864	6 Digits
1.23456e-15	896	896	
Counts are ± 30 cycles			

Figure 46. ASCII→float (cycles)

Format	Number	Single	Double
f9.2	.12345	672	704
	1	672	736
	1234.567	736	768
f17.10	.00000000001	576	576
	.0001	800	832
	1.23456789	864	992
	12345.6	864	1056
f27.20	1e-20	768	800
	1e-10	960	1088
	1	960	1344
	12345.6	928	1440
e9.2	1	936	960
	1e10	936	1040
	1e38	944	1408
e17.10	1	1104	1216
	1e10	1104	1240
	1e38	1112	1488
e27.10	1	1152	1560
	1e10	1160	1592
	1e38	1168	1648
g9.2	1e-37	1048	1512
	.01	1048	1088
	.5	736	776
	90	744	776
	1000	1056	1160
	1e38	1056	1528
g17.10	1e-37	1208	1760
	1e-10	1208	1352
	.01	1208	1344
	.5	904	1088
	1000	912	1040
	1e9	936	1056
	1e38	1232	1600
g27.20	1e-307	—	1936
	1e-37	1264	1944
	.01	1264	1776
	.5	960	1376
	1000	968	1400
	1e19	984	1488
	1e38	1312	1768
	1e308	—	1968
Counts are ± 30 cycles			

Figure 47. Float $\rightarrow$ASCII (cycles)

Applications Programmer's Section, continued

DATA TYPES

The WTL 3167 floating-point coprocessor provides compatibility with the formats specified in IEEE Standard 754, Version 10.0. Several number types are required to implement the standard. The types supported by the WTL 3167 are described below.

NORMALIZED NUMBERS (NRM)

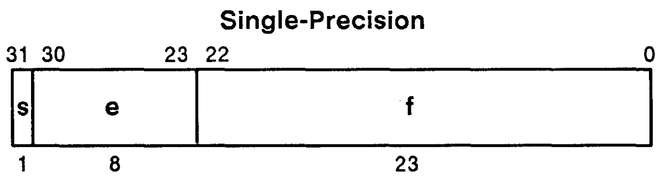
Most calculations are performed on normalized numbers. Single-precision normalized numbers have an exponent that ranges from binary 00000001 to binary 11111110 (1 to 254) and a normalized fraction field (the leftmost or hidden bit is a one). In decimal notation, this allows one to represent a range of both positive and negative numbers from roughly 10^{+38} to 10^{-38} with accuracy to seven decimal places. Double-precision numbers have an exponent ranging from one to 2,046 and a normalized fraction field.

INFINITY (INF)

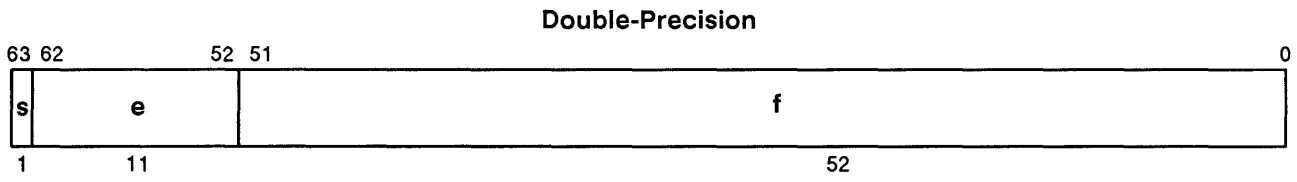
Infinity has an exponent of all ones and a fraction field equal to zero. Both positive and negative infinity are allowed.

ZERO

ZERO has an exponent of zero, a hidden bit equal to zero, and a value of zero in the fraction field. Both +0 and −0 are supported.



e	f	Value	Name
255	not 0	none	NaN (Not A Number)
255	0	$(-1)^s \times$ infinity	Infinity
1 .. 254	any	$(-1)^s \times 2^{e-127} \times (1.f)$	Normalized number
0	0	$(-1)^s \times 0$	Zero



e	f	Value	Name
2047	not 0	none	NaN (Not A Number)
2047	0	$(-1)^s \times$ infinity	Infinity
1 .. 2046	any	$(-1)^s \times 2^{e-1023} \times (1.f)$	Normalized number
0	0	$(-1)^s \times 0$	Zero

Figure 48. IEEE data types

NOT A NUMBER (NaN)

NaN is a special data format usually used as a flag for data flow control, for uninitialized variables, or to signify an invalid operation such as 0 times infinity. The format for a NaN is an exponent of all ones and a non-zero fraction.

DENORMALIZED NUMBERS (DNRM)

Denormalized numbers have a zero exponent and a denormalized (hidden bit equal to zero) non-zero fraction field. They represent numbers smaller than 2^{-127}(single-precision) or 2^{-1023}(double-precision).

ROUNDING OPTIONS

The WTL 3167 supports all four rounding modes of the IEEE standard: round to nearest, round toward zero, round toward plus infinity, and round toward minus infinity. Rounding may be biased or unbiased. Biased rounding introduces a small offset in the direction of the bias. Positive bias, negative bias, or a bias toward zero are specified in the IEEE format. Unbiased rounding rounds the result to the nearest representable number. In the case of a number exactly halfway between two representable numbers, the number is rounded toward the closest even number, resulting in half of the numbers rounding up and half rounding down, on average.

ROUND TO NEAREST (RN)

Rounds the result to the nearest representable value. If two numbers are equally near the result, the even number is chosen.

ROUND TOWARD ZERO (RZ)

Rounds the result to the value closest to but not greater than the magnitude of the result.

ROUND TOWARD PLUS INFINITY (RP)

Rounds the result to the value closest to but not less than the result.

ROUND TOWARD MINUS INFINITY (RM)

Rounds the result to the value closest to but not greater than the result.

IEEE CONSIDERATIONS

While the IEEE floating-point formats are supported by the WTL 3167, some features of the IEEE standard are not provided due to the design focus on high speed.

EXCEPTION HANDLING

The occurrence of an enabled exception causes an interrupt. Due to extensive instruction overlapping, the exact location of an exception is not maintained. In the debugging stage of a program it is possible to identify the instruction which caused the exception by performing a store context after every floating-point instruction and then testing the enabled exception bit.

The following exceptions are flagged by the WTL 3167:

Undefined Opcode Exception (UOE)

Whenever an illegal opcode is detected, the undefined opcode exception is set. On a read bus operation, for example, only store-type opcodes are allowed. If a read bus operation specifies any other instruction, such as MUL.S, then the undefined opcode exception bit is set.

Precision Exception (PE)

The precision exception (PE) flag of the accumulated exception field is set whenever there is a loss of accuracy. The coprocessor data paths compute results to higher precision than the number of mantissa bits that appear in the result. If any of the fraction bits less than the LSB was equal to one prior to rounding, then the PE bit will be set high. The precision exception will also be signaled if there is a partial or complete loss of significance in a float-to-fixed operation.

Applications Programmer's Section, continued

Overflow Exception (OE)

An overflow exception (OE) is generated when the result of a floating-point operation overflows the largest representable number. The result produced at the output is either infinity or the largest representable positive or negative number, depending upon the rounding mode as follows:

Largest positive normalized number	if ((RM or RZ) and the result is positive)
Largest negative normalized number	if ((RP or RZ) and the result is negative)
+Infinity	if ((RN or RP) and the result is positive)
−Infinity	if ((RN or RM) and the result is negative)

Overflow is also generated when converting floating-point-to-fixed point and the result overflows the 32-bit format.

Underflow Exception (UE)

When the result of an operation after rounding is less than the minimum normalized number in the destination format, UE is asserted and the result is flushed to zero. A result of exactly zero does not underflow.

Zero Divide Exception (ZE)

The WTL 3167 will assert a ZE exception when performing division on a normalized dividend and a zero divisor. The result is a properly signed infinity.

Invalid Operation Exception (IE)

IE is asserted if a NaN input or if an invalid operation occurs. The invalid WTL 3167 operations are $\infty \times 0$, 0/0, ∞/∞, subtraction of like infinities ($\infty - \infty$) and addition of opposite infinities $\infty+(-\infty)$. The result of any invalid operation is a NaN with the fraction and exponent of all ones. The sign bit is zero.

FAST MODE

The WTL 3167 always operates in Fast Mode: denormalized inputs to either the multiplier or ALU are flushed to zero as well as unnormalized outputs. The minimum normalized number has an exponent of one and a fraction field of zero. Zero has an exponent of zero and a fraction field of all zeros. This allows to represent numbers between the smallest normalized number and zero. These numbers are known as denormals (DNRM). Since denormals are very close to zero, most applications can substitute zero for a denormal without a significant loss of accuracy.

OPERATION STATUS AND RESULT

The following tables show the results which are obtained for various combinations of input data formats and rounding options. The format used in these tables is: (status) result. When OK is indicated for the status, no exception is flagged.

Source1	**Source2**				
	ZERO	DNRM	NRM	INF	NaN
NaN	(IE) NaN	(IE) NaN	(IE) NaN	(IE) NaN	(IE) NaN
INF	(OK) INF	(OK) INF	(OK) INF	(OK) INF (1) (IE) NaN (2)	(IE) NaN
NRM	(OK) NRM	(OK) NRM	(OE) (4) (OK) NRM (UE) ZERO (OK) ZERO	(OK) INF	(IE) NaN
DNRM	(OK) ZERO (3)	(OK) ZERO	(OK) NRM	(OK) INF	(IE) NaN
ZERO	(OK) ZERO (3)	(OK) ZERO (3)	(OK) NRM	(OK) INF	(IE) NaN

Notes:

1. +INF+INF → +INF
   -INF-INF → -INF
2. +INF-INF → NaN (invalid operation)
   -INF+INF → NaN (invalid operation)
3. +ZERO+ZERO → +ZERO (RN,RZ,RP,RM)
   -ZERO-ZERO → -ZERO (RN,RZ,RP,RM)
   +ZERO-ZERO → +ZERO (RN,RZ,RP)
   +ZERO-ZERO → -ZERO (RM)
   -ZERO+ZERO → +ZERO (RN,RZ,RP)
   -ZERO+ZERO → -ZERO (RM)
4. OVF will produce INF or maximum normalized number (MAX.NRM), depending upon the rounding mode:

+MAX.NRM	IF	[(RM,RZ)	AND	(RESULT IS +)]
-MAX.NRM	IF	[(RP,RZ)	AND	(RESULT IS -)]
+INF	IF	[(RN,RP)	AND	(RESULT IS +)]
-INF	IF	[(RN,RM)	AND	(RESULT IS -)]

Figure 49. Status and result output for add and subtract

Applications Programmer's Section, continued

Source1	**Source2**				
	ZERO	DNRM	NRM	INF	NaN
NaN	(IE) NaN	(IE) NaN	(IE) NaN	(IE) NaN	(IE) NaN
INF	(OK) INF	(OK) INF	(OK) INF	(IE) NaN	(IE) NaN
NRM	(ZE) INF	(ZE) INF	(OE) (4) (OK) NRM (UE) ZERO	(OK) ZERO	(IE) NaN
DNRM	(IE) NaN	(IE) NaN	(OK) ZERO	(OK) ZERO	(IE) NaN
ZERO	(IE) NaN	(IE) NaN	(OK) ZERO	(OK) ZERO	(IE) NaN

Note:

4. Refer to Note 4 on page 44.

Figure 50. Operation status and result output for divide

The following table shows the compare status for different input combinations; the compare status is encoded in the condition code field of the PCR.

Source1	Source2							
	NaN	-INF	-NRM	-DNRM	ZERO	+DNRM	+NRM	+INF
NaN	U	U	U	U	U	U	U	U
+INF	U	G	G	G	G	G	G	E
+NRM	U	G	G	G	G	G	0, 1, 2	L
+DNRM	U	G	G	E	E	E	L	L
ZERO	U	G	G	E	E	E	L	L
-DNRM	U	G	G	E	E	E	L	L
-NRM	U	G	0, 1, 2	L	L	L	L	L
-INF	U	E	L	L	L	L	L	L

U: Unordered
E: Source1 = Source2
L: Source1 < Source2
G: Source1 > Source2
0, 1, 2 may be: Source1 = Source2, Source1 < Source2, or Source1 > Source2

Figure 51. Status for floating-point compare

Applications Programmer's Section, continued

Source1	Source2/Destination	Status	Comments
7FFFFFFF	41DFFFFF FFC00000	OK	Largest positive integer
00000001	3FF00000 00000000	OK	+1
00000000	00000000 00000000	OK	ZERO
FFFFFFFF	BFF00000 00000000	OK	−1
80000000	C1E00000 00000000	OK	Largest negative integer

Figure 52. Integer to double-precision conversions (I32→F64)

Source1	Source2/Destination	Status	Comments
7FFFFFFF	4F000000	OK	Largest positive integer
7FFFFFC0	4F000000	PE	Inexact
7FFFFF80	4EFFFFFF	OK	Exact
00000001	3F800000	OK	+1
00000000	00000000	OK	ZERO
FFFFFFFF	BF800000	OK	−1
80000080	CEFFFFFF	OK	Exact
80000040	CF000000	PE	Inexact
80000000	CF000000	OK	Largest negative integer

Figure 53. Integer to single-precision conversions (I32→F32)

Systems Programmer's Section

The system software is responsible for mapping logical addresses to the physical address space of the WTL 3167, detecting the presence of the WEITEK coprocessor, handling exceptions, saving the coprocessor registers when switching between tasks, and emulating the device when it is not present. In non-multi-tasking environments like MS-DOS, only address mapping, and presence detection need to be performed.

SETTING-UP WTL 3167 ADDRESSING

The Operating System must provide a mechanism to map logical addresses into the proper WTL 3167 physical addresses.

MS-DOS ENVIRONMENT

Ordinarily, the WTL 3167 memory space is inaccessible in real mode, since Intel intended only the first megabyte of the 386 memory space to be used. However, there is an anomaly of real-mode memory addressing that allows an extra 65520 bytes of the memory space to be accessed, which is enough to accommodate the WTL 3167. The anomaly occurs when a segment register is loaded with the value FFFF hex, and an offset of 10 hex or greater is provided for a memory address. After multiplying the segment register value by 16 and adding the offset, an address exceeding the 1-megabyte boundary is obtained. On the original 8086/8088, the address wraps around to zero. On the 386, the address extends into the second megabyte of the memory space. This allows a real-mode program to access linear addresses from 100000 hex to 10FFEF hex. The 386's paging mechanism can map those linear addresses to physical addresses in the WTL 3167's memory space.

The paging must be set up by an initialization program run at boot time. The program must enter the 386's Virtual 8086 mode, set up paging tables and segment descriptor tables, address memory according to those tables, an then go back to real mode. The paging mechanism must map the first megabyte of memory to itself, and must map the addresses from 100000 hex to 10EFFF hex to the WTL 3167 space at 0C0000000 hex through 0C000EFFF hex.

If there is an Extended Memory Manager, the page mapping should be handled at the same time. Because there exist programs which rely on the wraparound of the addresses greater than one megabyte, the Extended Memory Manager should provide for the ability to dynamically turn the WTL 3167 mapping on and off. The steps to perform memory-mapping in a Virtual 8086 environment are explained in detail in the Memory Management chapter of the *Intel 80386 Programmer's Reference Manual.*

Once the paging is set up, the WTL 3167 space can be accessed starting at 0FFFF:10 hex.

DOS PROTECTED MODE ENVIRONMENT

In the MS-DOS protected mode environment the application program runs in 386 protected mode to execute native 386 code and/or access the larger memory space. Currently there are three tools available for running programs in protected mode under MS-DOS: *RUN386* by Phar Lap, *X-AM* by IGC, and OS 386 by AI Architects. The *RUN386* program sets up the addressing for the WTL 3167, by pointing the fs register to a segment containing the WTL 3167 memory space. The WTL 3167 space starts at offset 0 hex within the fs segment. Under *X-AM*, the processor assumes a flat segmentation model: all segment registers are set to zero, and the entire 386 memory space is accessed via 32-bit offset values. The WTL 3167 memory space is re-mapped from 0C0000000 hex to the location 0FFC00000 hex.

Since all segment registers point to the same zero value, no segment override bytes are necessary when running under *X-AM*. The default registers ds, es and ss will always suffice to access the WTL 3167's space.

UNIX AND XENIX ENVIRONMENTS

UNIX and XENIX provide a flat memory space, with all 386 segment registers pointing to zero, and the entire memory space addressed through 32-bit offsets. UNIX handles the page re-mapping of the WTL 3167 memory space, so that the applications program can immediately access the WTL 3167 starting at offset 0FFC00000 hex.

Systems Programmer's Section, continued

COPROCESSOR PRESENCE DETECTION

Many application programs will need to test for the existence of the WEITEK coprocessor (either WTL 3167 or WTL 1167). If an application program needs to decide whether to run on the WEITEK coprocessor or the 80387, this information is necessary.

MS-DOS ENVIRONMENT

In the MS-DOS real mode environment, a simple program in the ROM BIOS, or resident in main memory must detect the presence of the WEITEK coprocessor, and modify the interrupt 11 hex routine so that bits 23 and 24 of the value returned in register eax by the interrupt 11 hex call are set if the WEITEK coprocessor is present.

To detect the presence of the WEITEK coprocessor systems programmers will use a simple detection routine consisting of a software sequence that loads a coprocessor register with a specific data pattern and then reads it back. The code fragment in figure 54, based on the MS-DOS macros offered by WEITEK, can be used to detect the WEITEK coprocessor presence and modify the interrupt 11 hex routine. It is important to note that this code assumes that the page tables for real mode addressing have been set up. It also takes advantage of the fact that a reference to memory that does not exist will eventually return some undefined result and the system will not hang. It finally assumes that the physical address of the WEITEK coprocessor is the standard address of 0C0000000 hex.

If the hardware designer has connected the PRES– signal of the WEITEK coprocessor to an I/O port, the systems software designer can determine the coprocessor presence by reading such I/O port. This method is simpler than the previous one but it is system dependent. It may be necessary to change the I/O port address for each machine.

Once the operating system has detected the presence of the WEITEK coprocessor and modified the interrupt 11 routine, available DOS compilers and applications can detect the presence of the WEITEK coprocessor by calling Interrupt 11 hex and checking the eax bits 23 and 24 status as shown in figure 55.

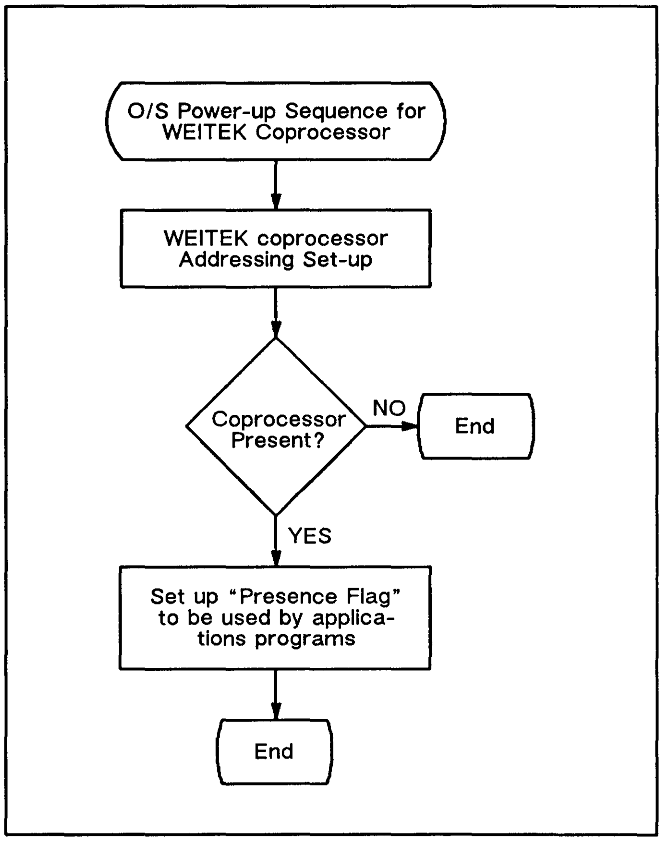

O/S Power-up Sequence for WEITEK Coprocessor
WEITEK coprocessor Addressing Set-up
Coprocessor Present?
NO
End
YES
Set up "Presence Flag" to be used by applications programs
End

Figure 54. Operating systems power-up sequence for WTL 3167

```
; see if WEITEK coprocessor is present
XOR       EAX, EAX
INT       11h
AND       EAX, 11 shl 23
JNZ       short J3167
K3167:    ; WEITEK coprocessor not present
J3167:    ; WEITEK coprocessor is present
```

Figure 55. Compiler test for presence of WTL 3167

MS-DOS PROTECTED MODE ENVIRONMENT

The MS-DOS protected mode environment (RUN386, X-AM, or OS 386) is responsible for detecting the presence of the WEITEK coprocessor. Code similar to that proposed in figure 56, properly modified for protected mode addressing, can be used by MS-DOS protected mode development environment manufacturers to detect the presence of the WEITEK coprocessor.

Once the environment has detected the presence or absence of the coprocessor it must provide a way to communicate it to application programs. Such method is specific to the environment. Clearly the INT 11h mechanism cannot be used in protected mode. For Phar Lap-based environments, for example, identical ds and fs segment registers indicate to the application that the WEITEK coprocessor is present. For other environments the reader should consult the manufacturer's documentation.

UNIX AND XENIX ENVIRONMENTS

UNIX and XENIX environments detect the presence of the WEITEK coprocessor using routines similar to that shown in figure 56. In both UNIX and XENIX environments a program can detect the presence of the WEITEK coprocessor through system call "sysi86". See the appropriate system documentation for the name of the system call and the parameter definitions.

Systems Programmer's Section, continued

```
; see if WEITEK coprocessor is present
; (this code assumes that page tables for real mode addressing have already been set up)
; load FFFFh into fs segment register
MOV     fs, FFFFh
; save contents of memory which may change if WEITEK coprocessor is not present
MOV     ECX, fs:0404h
MOV     EDX, fs:0408h
; read register ws1 into eax
WFST    EAX, ws1
; write the data now in EAX into WEITEK coprocessor register ws2
WFLD    ws2, EAX
; complement data in EAX, save it in EBX, and write it back into register ws1
NOT     EAX
MOV     EBX, EAX
WFLD    ws1, EAX
; read the two WEITEK coprocessor registers ws1 and ws2, and compare them to EBX
WFST    EAX, ws1
CMP     EAX, EBX
WFST    EAX, ws2
; restore memory which may have changed
WFLD    ws1, EAX
WFLD    ws2, EDX
; restore Interrupts
STI
; branch if either register does not compare
JNZ     short i0init
NOT     EAX
CMP     EAX, EBX
JNZ     short i0init
; if the WEITEK coprocessor is present the system must modify the interrupt 11h routine so that bits
; 23 and 24 of the value returned by interrupt 11h in EAX is set (See Note). Application software
; will then use this mechanism to determine whether the WEITEK coprocessor is present.
MOV     di, offset Handlerjump
MOV     dword ptr [di-4], 11 shl 23
i0init:                         ; WEITEK coprocessor is not present
```

Note: the code that modifies interrupt 11h assumes that the interrupt handler has been previously loaded as follows:

```
Handler:
MOV     EAX, 0
Handlerjump:
JMP     far ptr original        ; Jump to original interrupt 11h handler routine
```

Figure 56. Test for presence of WEITEK coprocessor (WTL 3167 or WTL 1167)

INITIALIZATION

Multitasking operating systems and application programs must initialize the WEITEK coprocessor. The code in figure 58, written using WEITEK macros, will suffice to initialize any WEITEK coprocessor: WTL 3167 old or new, WTL 1167, WTL 1167 type A.

The rounding mode, the exception mask field and the accumulated exception field of the PCR need to be initialized as well. The instruction in figure 57, for example, will set round to nearest rounding mode and will mask and clear all exceptions.

```
; initialize exception masks and rounding mode
WFLDCTX  003FF0000h
```

Figure 57. Exception mask and rounding mode initialization

```
WFLDCTX  B8000000h       ; load B8000000h int PCR
WFSTRL   EAX             ; store revision level
CMP      ah, 00h
JNE      short j1init
k1init:
WFLDCTX  016000000h      ; initialize Multiplier and ALU units flowthrough timers in
                         ; WTL 1167
JMP      short i1init
j1init:
WFLDCTX  056000000h      ; initialize Multiplier flowthrough timer in WTL 1167 type A
WFLDCTX  098000000h      ; initialize ALU flowthrough timer in WTL 1167 type A
i1init:
; regardless of the coprocessor type load the following remaining power-up sequence
WFLDCTX  064000000h
WFLDCTX  0A0000000h
WFLDCTX  030000000h
```

Figure 58. Initializing the WEITEK coprocessor

Systems Programmer's Section, continued

EXCEPTION HANDLING

When an enabled exception occurs, the WTL 3167 signals an interrupt to the host processor. The 80287/80387 and the WTL 3167 interrupt requests are "ORed" to generate the exception interrupt (see page 8). Whenever an interrupt occurs, the operating system must first check both the WTL 3167 and the 80287/80387 to assess which device flagged the exception. To handle the WTL 3167 exception interrupt the operating system must first clear the interrupt, in order to allow processing to continue, and then transmit the interrupt information to the executing program. The operating system can simply notify the executing program of a problem, expecting the application program to identify and correct the problem, or it can identify the problem and then pass the information to the application program. In the first case it should clear the interrupt by setting the appropriate bits in the enable exception byte, but leave untouched the accumulated exception byte.

This allows the executing program to determine exactly which exceptions occurred by reading the context register itself. In the second case, the operating system clears the interrupt by storing the value of the accumulated exception byte and then clearing it. The content of the accumulated exception byte is then passed to the application program.

MS-DOS application programs handle IRQ13 interrupts by trapping INT 75. After resolving WTL 3167 exceptions, the routine clears the exception byte and chains to the INT 75 vector. The INT 75 service routine clears the interrupt controllers and invokes the Non Maskable Interrupt (NMI) handler (for compatibility with 8088/8086 software).

CONTEXT SWITCHING

In a multi-tasking environment, such as UNIX and XENIX, the operating system must save the context of the WTL 3167 when switching between two processes. Saving the WTL 3167 simply means saving the thirty-two registers in the register file and the context register. A block move is very effective in saving the register file. Restoring the WTL 3167 context is simple too. The thirty-two registers must be reloaded as well as the context register. It is also required that the operating system repeat the coprocessor initialization described in figures 57 and 58 in case another program had loaded the WTL 3167, rounding mode, exception mask, and accumulated exception field of the PCR with inappropriate values. UNIX System V.3 and XENIX for the 80386 handle context switching for the WTL 3167.

COPROCESSOR EMULATION

If emulation of the WTL 3167 is needed, then it is the operating system's responsibility to provide it. When the WTL 3167 is not present and an address specifying a coprocessor instruction is broadcast by the 80386, the operating system must identify the fault and call the emulator. The emulator needs to decode the address in order to identify the floating-point instruction which it specifies, and it must then execute the instruction with the system's available resources. The emulator must duplicate all of the WTL 3167's internal registers and properly update them after each instruction. UNIX System V.3 already incorporates a complete WTL 3167 emulator. Customers who intend to incorporate WTL 3167 emulation in other operating systems should contact WEITEK for more details.

WTL 1167 and WTL 3167 Compatibility

This section describes the hardware and software differences between the WTL 3167 and the WTL 1167 coprocessor daughter board.

HARDWARE COMPATIBILITY

The single-chip WTL 3167 is pin for pin compatible with the WTL 1167 and will fit into the standard 121-pin extended math coprocessor socket. The WTL 1167 coprocessor daughter board features a socket for the 80387, allowing both the WEITEK and the Intel coprocessors to co-exist in the same system. Hardware developers can offer the option of using both the WTL 3167 and the 80387 coprocessors by featuring two separate sockets on the system mother board, or by using a small daughter board that accommodates both coprocessors. Figure 5 shows the physical dimensions of the WEITEK daughter board that accommodates both the 80387 and the WTL 3167.

The WTL 3167 DC power consumption is less than one fifth that of the WTL 1167 daughter board. The 16 and 20 MHz WTL 3167 AC specs are upward compatible with those of the WTL 1167 daughter board. The WTL 3167 is available in faster speed grades than the WTL 1167. For more details on the coprocessor DC and AC specifications, the reader should refer to pages 9 to 11. The WTL 3167 AC specifications match the new AC specifications for the Intel 80386 microprocessor.

APPLICATION SOFTWARE COMPATIBILITY

The WTL 3167 is upward object-code-compatible from the WTL 1167. The application programs and all of the software tools available for the WTL 1167 coprocessor daughter board will run as is on the WTL 3167. The WTL 3167 will respond as a faster WTL 1167. For more details on the single-chip instruction execution times, refer to pages 37 to 40.

The WTL 3167 features some new instructions that will trigger an Invalid Opcode exception, if used with the WTL 1167. The new instructions include: square root, reverse subtract, and double-precision multiply accumulate.

SYSTEM SOFTWARE COMPATIBILITY

Addressing, initialization, presence detection, exception handling, context switching, and coprocessor emulation for the WTL 3167 are the same as they are for the WTL 1167. Therefore, the WTL 3167 works in all of the operating system environments that support the WTL 1167 coprocessor daughter board.

Ordering Information

COPROCESSOR

Part Description	Temperature Range	Order Number
16 MHz WTL 3167 Coprocessor	T_{CASE} = 0 to 85° C	3167-016-GCU
20 MHz WTL 3167 Coprocessor	T_{CASE} = 0 to 85° C	3167-020-GCU
25 MHz WTL 3167 Coprocessor	T_{CASE} = 0 to 85° C	3167-025-GCU

Figure 59. WTL 3167 Coprocessor ordering information

COPROCESSOR BOARD

Customers ordering the coprocessor along with the small daughter board shown in figure 5, should refer to the order numbers below.

Part Description	Temperature Range	Order Number
16 MHz WTL 3167 Coprocessor Board	T_{CASE} = 0 to 85° C	3167-016-BRD
20 MHz WTL 3167 Coprocessor Board	T_{CASE} = 0 to 85° C	3167-020-BRD
25 MHz WTL 3167 Coprocessor Board	T_{CASE} = 0 to 85° C	3167-025-BRD

Figure 60. WTL 3167 Coprocessor board ordering information

Revision Summary

This table lists many of the most major changes since the September, 1986 printing of this data sheet. The data sheet has undergone a complete transformation since then. It is now more accurate, more complete, and much longer. Few, if any, sections from the old data sheet exist unchanged in the new one.

	Change	Description
1.	Specifications	Revised, page 10–11
2.	Software Tools Overview	Revised, page 20
3.	Instruction Encoding	Revised, page 32
4.	Ordering Information	Revised, page 55

WTL 3167 FLOATING-POINT COPROCESSOR

PRELIMINARY DATA

September 1988

For additional information on WEITEK products, please fill out the form below and mail.

Name ______________________ Title ______________________

Company ______________________ Phone ______________________

Address ______________________

Comments ______________________

I am currently involved in a design with the following Weitek products ______________________ and wish to be added to your design data base to insure that I receive status updates.

APPLICATION:

☐ ENGINEERING WORKSTATIONS	☐ SCIENTIFIC COMPUTERS
☐ GRAPHICS	☐ OTHER __________
☐ PERSONAL COMPUTERS	

Check the products on which you wish to receive data sheets: ☐ Have a sales person call

ATTACHED PROCESSORS	COPROCESSORS	BUILDING BLOCKS		
☐ XL-SERIES OVERVIEW	☐ 1167	☐ 2264/2265	☐ 1066	☐ 2516
☐ XL-8200 OVERVIEW	☐ 1164/1165	☐ 3132/3332	☐ 2010	☐ 2517
	☐ 3164/3364	☐ 1232/1233	☐ 2245	
	☐ 3167			

WEITEK use: Rec'd Out TPT Source: DS

Status

WEITEK WTL 3167
Please Comment On The Quality Of This Data Sheet.

Even though we have tried to make this data sheet as complete as possible, it is conceivable that we have missed something that may be important to you. If you believe this is the case, please describe what the missing information is, and we will consider including it in the next printing of the data sheet.

Fold, Staple and Mail to Weitek Corp.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1374 SUNNYVALE, CA

POSTAGE WILL BE PAID BY ADDRESSEE

WEITEK Corporation
1060 E. Arques Ave.
Sunnyvale, CA 94086-BRM-9759

ATTN: Ed Masuda

WEITEK

WEITEK'S CUSTOMER COMMITMENT:

Weitek's mission is simple: to provide you with VLSI solutions to solve your compute-intensive problems. We translate that mission into the following corporate objectives:

1. To be first to market with performance breakthroughs, allowing you to develop and market systems at the edge of your art.

2. To understand your product, technology, and market needs, so that we can develop Weitek products and corporate plans that will help you succeed.

3. To price our products based on the fair value they represent to you, our customers.

4. To invest far in excess of the industry average in Research and Development, giving you the latest products through technological innovation.

5. To invest far in excess of the industry average in Selling, Marketing, and Technical Applications Support, in order to provide you with service and support unmatched in the industry.

6. To serve as a reliable, resourceful, and quality business partner to our customers.

These are our objectives. We're committed to making them happen. If you have comments or suggestions on how we can do more for you, please don't hesitate to contact us.

[illegible]

Art Collmeyer
President

Headquarters	**Domestic Sales Offices**		**European Sales Headquarters**	**Japanese Representative**
Weitek Corporation	Weitek Corporation	Corporate Place IV	Greyhound House, 23/24 George St.	4-8-1 Tsuchihashi
1060 E. Arques Avenue	1060 E. Arques Avenue	111 South Bedford St.	Richmond, Surrey, TW9 1JY	Miyamae-Ku
Sunnyvale, CA 94086	Sunnyvale, CA 94086	Suite 200	England	Kawasaki, Kanagawa-Pre
TWX 910-339-9545	TWX 910-339-9545	Burlington, MA 01803	TELEX 928940 RICHBI G	213 Japan
WEITEK SVL	WEITEK SVL	FAX (617) 229-4902	FAX 011-441 940 6208	FAX 044-877-4268
FAX (408) 738-1185	FAX (408) 738-1185	TEL (617) 229-8080	TEL 011-441 549 0164	TEL 044-852-1135
TEL (408) 738-8400	TEL (408) 738-8400			