WTL 3132/WTL 3332/XL-3132
32-BIT FLOATING POINT DATA PATH

October 1988

Features

32-BIT FLOATING POINT PROCESSOR

Single-precision floating point multiplier/ALU

Four-port 32×32 register file

IEEE floating point format

Low power, high integration CMOS

FULL FUNCTION

Add, subtract, multiply, multiply/accumulate

Divide look-up table

Type conversion to and from two's complement integer

Three-address (rc := ra + rb) architecture

Flexible I/O options

HIGH PERFORMANCE

100 and 120 ns cycle times

Up to 20 MFLOPS throughput (1 MAC/cycle)

Low latency (3-cycle register-to-register operation)

High I/O bandwidth (up to 200 Mbytes/sec)

MULTI-PURPOSE

For maximum throughput, use the three-port WTL 3332

For maximum design flexibility, use the WTL 3132 or WTL 3332 as microprogrammable building blocks

For high level language support, use the XL-3132 as the XL-8032 floating point coprocessor

Description

The WTL 3132 and WTL 3332 are single-precision floating point data paths. Each includes a pipelined multiplier/accumulator and a four-port register file with thirty-two 32-bit registers.

The WTL 3132/WTL 3332 are suited to a wide range of systems that need high numeric processing performance. They may be adopted as the floating point unit for a general-purpose processor, used as building blocks for application-specific data paths or even connected together to create vector or array processors.

The WTL 3132 has a single bi-directional 32-bit input/output port. It is designed to be used as a floating point coprocessor or accelerator. The WTL 3332 has three 32-bit ports; one bi-directional input/output port, one input port and one output port. It should be used in applications which require multiple high-bandwidth buses.

The XL-3132 may also be used with the WEITEK XL-8136 program sequencing unit (PSU) and XL-8137 integer processing unit (IPU) to create a fast, general-purpose numeric processor, the XL-8032. Full development system support, including FORTRAN and C compilers, is available for the XL-Series of processors. The XL-3132 is functionally identical to the WTL 3132.

Both devices are manufactured in low power CMOS and are available in standard pin grid array (PGA) packages. The WTL 3132 is supplied in a 144-pin PGA and the WTL 3332 in a 168-pin PGA.

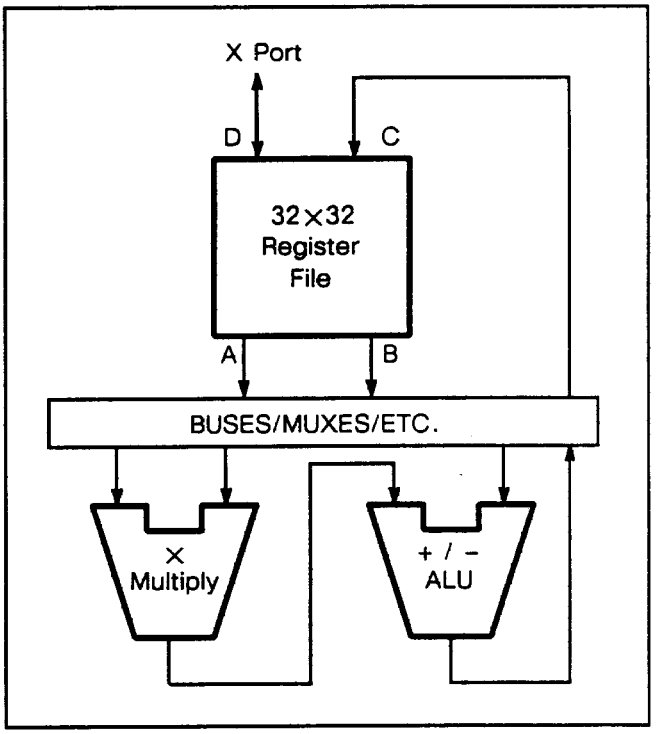


Figure 74. WTL 3132/WTL 3332 core functions

Architecture

MULTIPLIER/ACCUMULATOR

The core of both the WTL 3132 and WTL 3332 is the multiplier/accumulator pipeline. Its first stage can multiply two operands together. The next stage can add or subtract another operand. Finally, the result is rounded and returned to a register and/or output port.

Multiply, add, subtract, and multiply/accumulate operations are performed in the multiplier/accumulator. They all operate on data that conforms to the IEEE single-precision floating point format.

Each operation takes three cycles, but, because the multiplier/accumulator is pipelined, a new operation can be started on every cycle. At any time, three independent operations may be at different stages in their execution.

Rounding, conversion between floating point, and two's complement integer formats, and other miscellaneous functions are supported in the accumulator.

REGISTER FILE

Operands and results of the multiplier/accumulator may be stored in the four-port register file. This file contains thirty-two registers, each of which may store a 32-bit value.

The four ports allow the register file to supply two operands to the multiplier/accumulator, store its result back to a register, and perform an input/output transfer—all in the same cycle.

INPUT/OUTPUT PORTS

The external I/O ports are all 32 bits wide. They can each transfer a data value on every cycle.

The WTL 3132 has one bi-directional external port; the X port. It can load and store data to and from the register file, and it can transfer data directly to and from the multiplier/accumulator.

The WTL 3332 has three external ports; the X port, the Y port, and the Z port. The X port is the same as the WTL 3132's X port. The Y port feeds input operands directly to the multiplier/accumulator. The Z port outputs results directly from the multiplier/accumulator. These additional ports help to avoid the bottlenecks usually associated with I/O-intensive algorithms.

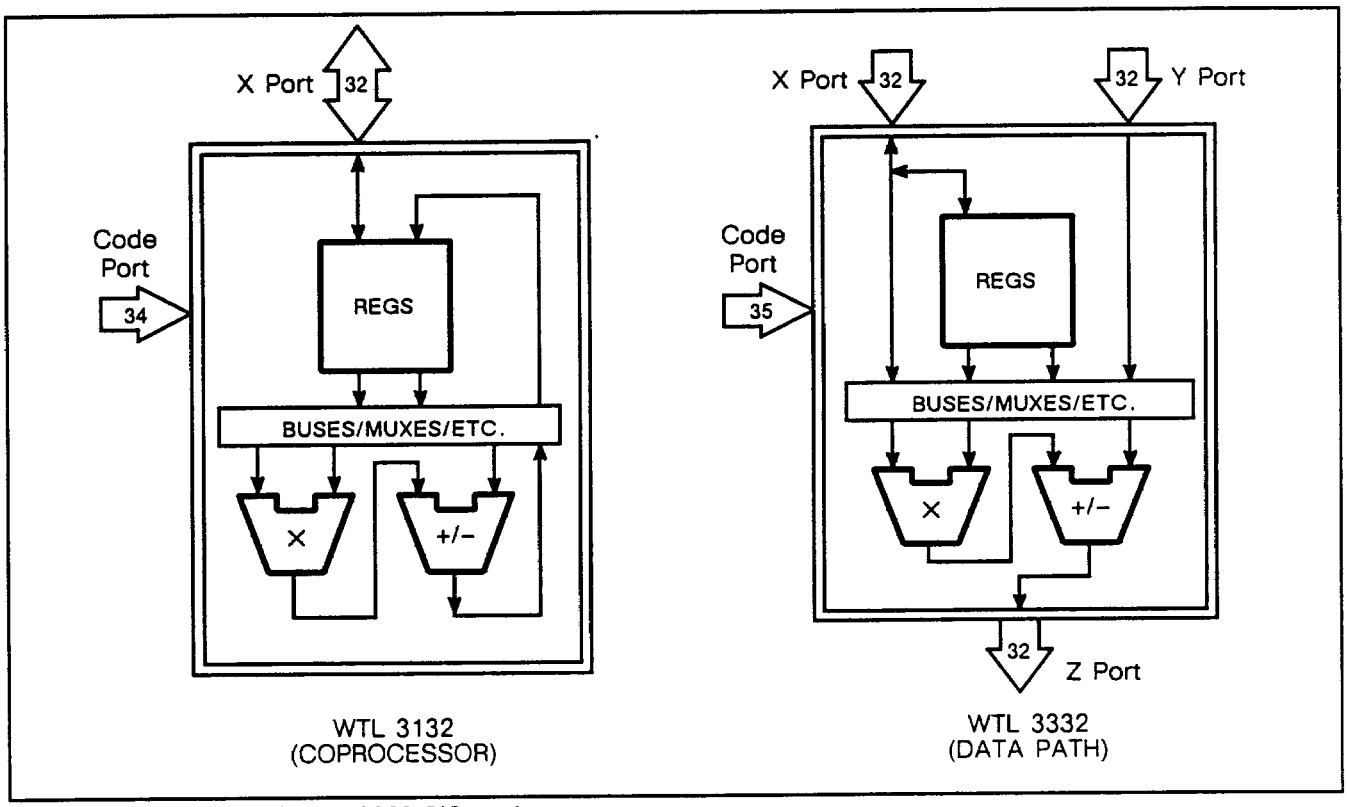


Figure 75. WTL 3132/WTL 3332 I/O options

Architecture, continued

TEMPORARY REGISTERS

Three 32-bit temporary registers are provided to store intermediate results. They make it possible to perform operations of the form

$x = x \pm (y \times z)$

in a single cycle.

DIVIDE LOOK-UP TABLE

Support for divide operations is provided by an on-chip look-up table. It returns an approximation for the inverse of a value which may then be refined by iterative multiply/accumulate operations. Division is accomplished by multiplying the dividend by the inverse of the divisor. This complete divide operation takes eighteen cycles; other operations may be interleaved without a performance penalty.

INSTRUCTIONS

An instruction is latched into the code port on every cycle. It specifies operand sources, a result destination and all of the steps that will create this result during the next three cycles. Condition codes and exceptions may be generated by each operation as the result is written back to the register file.

Four five-bit fields provide addresses for the register file. They each select a source or destination for one of the register ports. The three-bit function field specifies the type of multiplier/accumulator operation. The 2-bit I/O control field directs data transfer at the external X port. Other fields select the route taken by the data during the the operation.

A Mode Register controls data routing options that rarely change, selecting between a number of I/O timing options, and supporting upward-compatibility with previous versions of the WTL 3132/WTL 3332.

XL-SERIES COMPATIBILITY

The XL-3132 may be used with the WEITEK XL-8136 program sequencing unit (PSU) and XL-8137 integer processing unit (IPU) to create the XL-8032 processor.

The XL-3132 floating point unit (FPU) shares a 64-bit instruction word with the IPU and PSU. The IPU and FPU also share the 32-bit-wide data bus.

The XL-3132 responds to the NEUT–, STALL– and ABORT– signals used to control branching and "wait states" within the XL-8032. It communicates its status to the PSU with the floating point condition (FPCN) and exception (FPEX) lines. See Appendix A for more about the XL-Series.

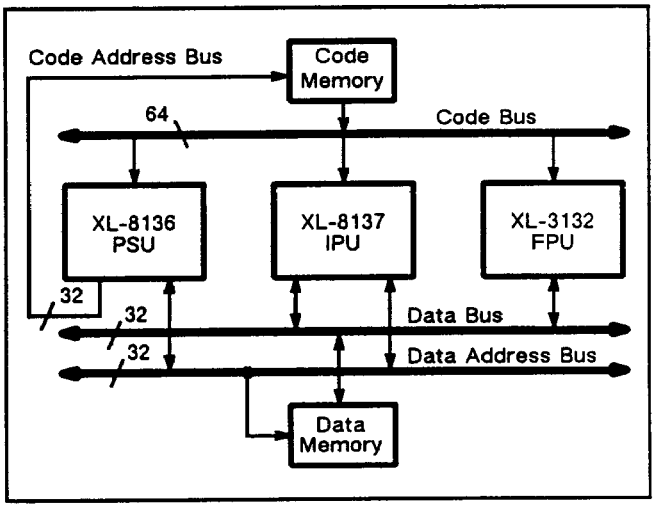


Figure 76. XL-8032 block diagram

Signal Description

X PORT

The 32-bit $X_{31..0}$ port is a bi-directional data bus. Input data is sampled on the rising edge of CLK (or, if Double-Pump Mode is enabled, both on the rising and falling edges of CLK). Data transfers are controlled by the $IOCt_{1..0}$ field in the instruction word. The X port may be set to a high impedance state by the OEX- signal. Active high.

Y PORT

The 32-bit $Y_{31..0}$ port is a data input bus. Input data is sampled on the rising edge of CLK (or, if the Y port Late Input Mode is enabled, on the falling edge of CLK). The Y port is only available on the WTL 3332. Active high.

Z PORT

The 32-bit $Z_{31..0}$ port is a data output bus. The output data is modified on every cycle. The Z port is only available on the WTL 3332. It may be set to a high impedance state by the OEZ- signal. Active high.

C PORT

The 35-bit $C_{34..0}$ port is used as a code input bus. Instructions are latched the rising edge of CLK. Active high. C_{34} is only available on the WTL 3332 (see figure 40).

OEX-

X port output enable input. OEX- asynchronously disables the X port when high. Active low.

OEZ-

Z port output enable input. OEZ- asynchronously disables the Z port when high. Active low.

FPEX(-)

Floating point exception output. FPEX signals the occurrence of an enabled exception (overflow). Polarity is selectable by mode bit M_8.

FPCN

Floating point condition output. FPCN signals the occurrence of a condition as specified in the $Encn_{1..0}$ field of an instruction. Active high.

ZERO

Zero condition output. Indicates that the result of an operation is exactly equal to zero. Controlled by the $Encn_{1..0}$ field of an instruction. Active high.

NEUT-

Neutralize input. Cancels the effect of the current instruction. Typically used during delayed branches and interrupt response routines (see page 27). Latched on the cycle following the instruction to be cancelled. Active low.

STALL-

Stall input. Cancels the effect of the next instruction. Typically used as a "not ready" line from the code store (see page 28). Latched on the same cycle as the potentially invalid instruction. Active low.

ABORT-

Abort input. Cancels the effect of both the current and next instructions. Typically used as a "not ready" line from the data store (see page 29). Latched on the same cycle as the next instruction. Active low.

CLK

Clock input. TTL compatible.

VDD

All VDD pins must be connected to 5.0V.

GND

All GND pins must be connected to system ground.

NOTE: Signals denoted by "-" are active low.

Block Diagram

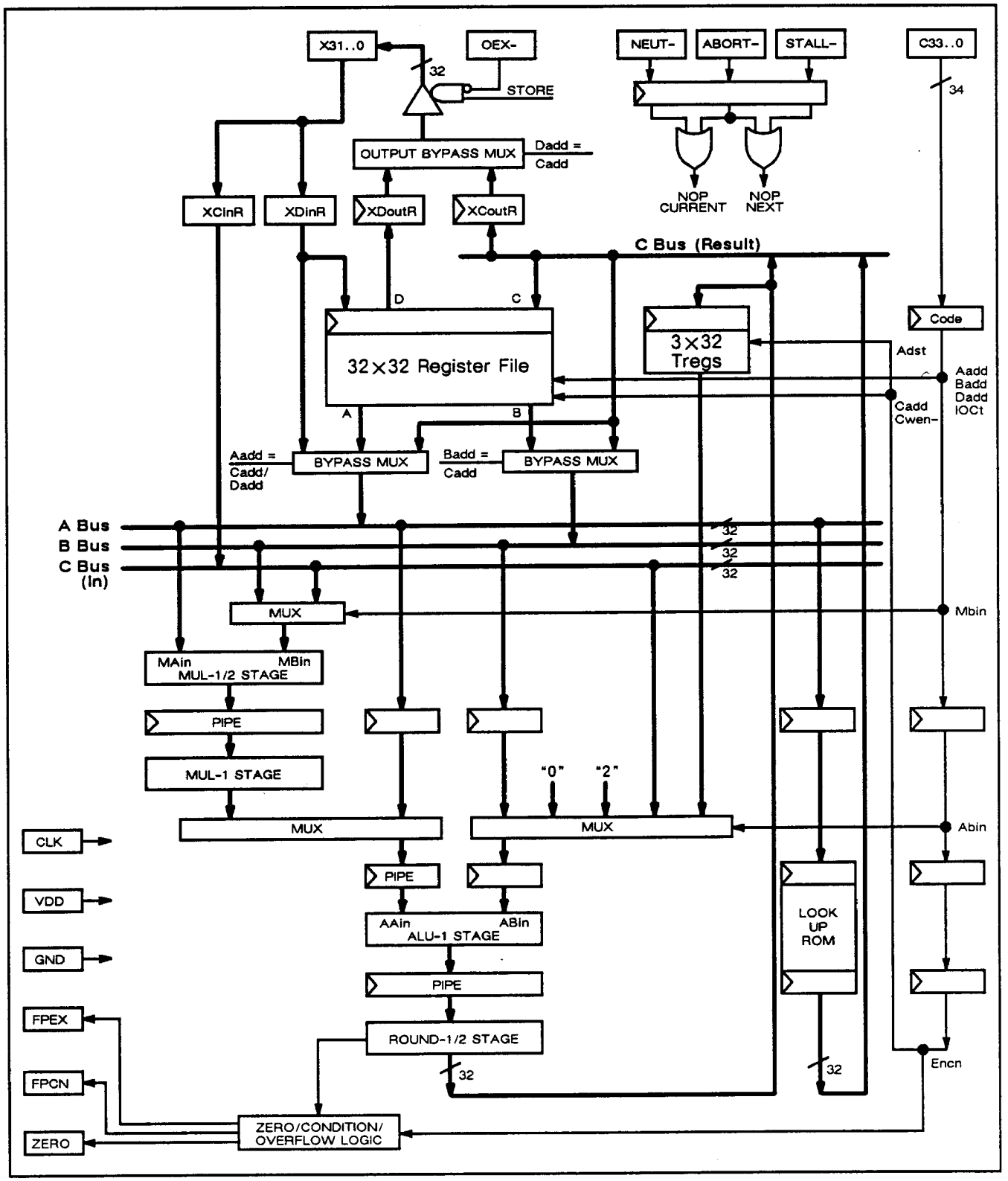


Figure 77. WTL 3132 block diagram

Block Diagram, continued

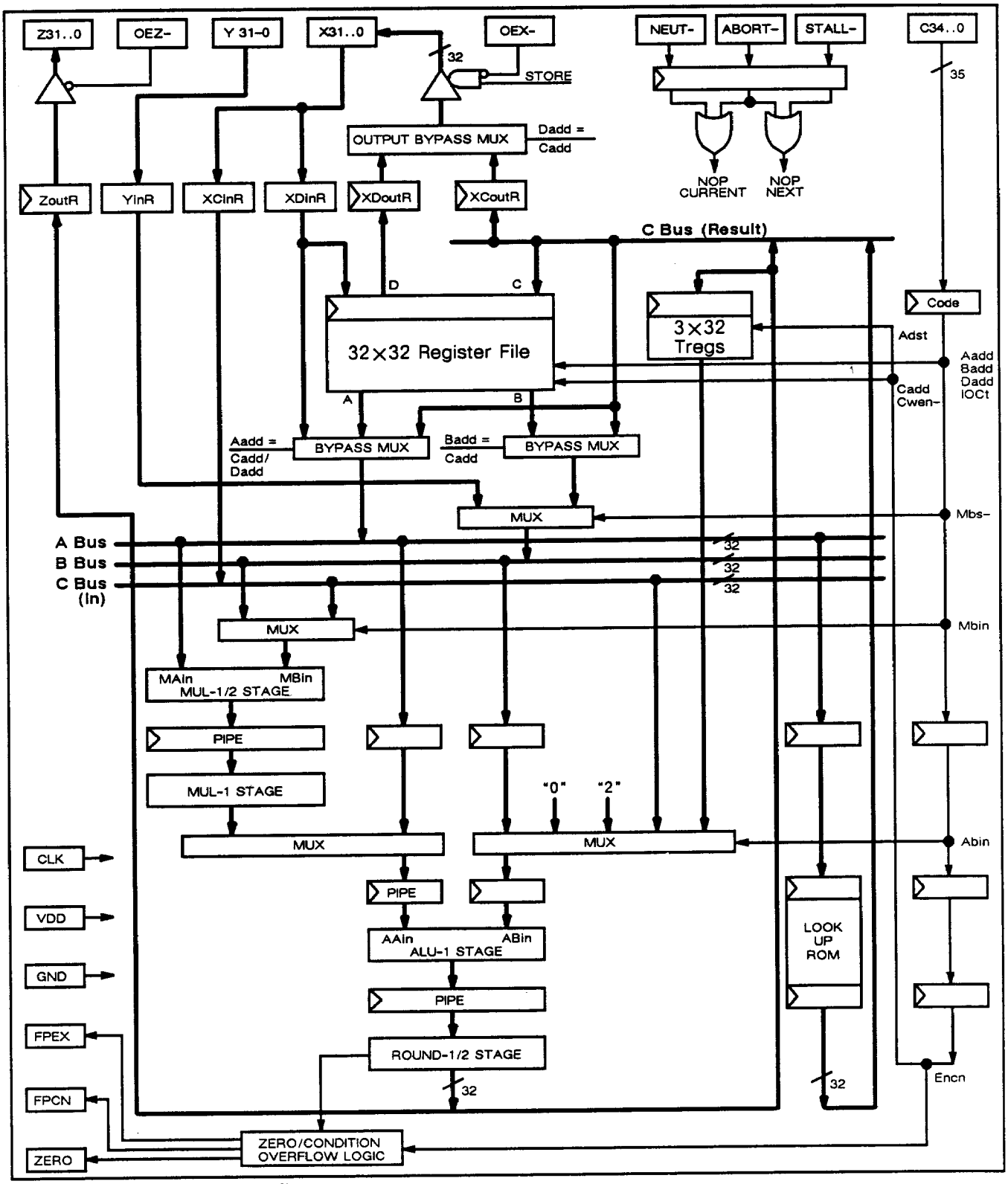


Figure 78. WTL 3332 block diagram

Register File

The WTL 3132 and WTL 3332 each have thirty-two 32-bit general-purpose registers. Each register can store either a single-precision IEEE value or a two's complement integer value.

PORTS

The register file has four ports, A, B, C and D. The A and B ports are read-only, the C port is write-only, and the D port is bi-directional. Each port can transfer a 32-bit data word on every clock cycle.

The A and B ports may be used to supply operands to the multiplier/accumulator and the divide look-up table. The C port receives the result of a previous operation. The D port communicates data between the register file and the external X port.

This organization allows I/O transfers to proceed in parallel with calculation, maximizing system performance.

REGISTER SELECTION

The registers that are to take part in each transfer are selected by the instruction word. The instruction format allows a register address to be supplied for each port. They are provided in the Aadd, Badd, Cadd and Dadd fields of the instruction. These fields are five bits in length, allowing each address to specify any of the thirty-two registers.

An instruction supplies the Aadd, Badd, and Dadd addresses to the register file during its first cycle and the Cadd address during its fourth cycle. This way a single instruction specifies all of the stages of an operation from initial source to ultimate destination.

It is possible for a register to be selected by more than one field in the same cycle, in which case the following rules apply:

1. If only read operations are to be performed on the register in question, then its value is copied to all of the necessary ports.
2. If two ports (C and D) attempt to write into the same register on the same cycle, the contents of the register will be left in an undefined state. Such contention should be avoided.
3. If a register is to be both read and written on the same cycle, its old value will be read before it is updated to the new value, unless one of the Bypass Modes is activated (see page 16).

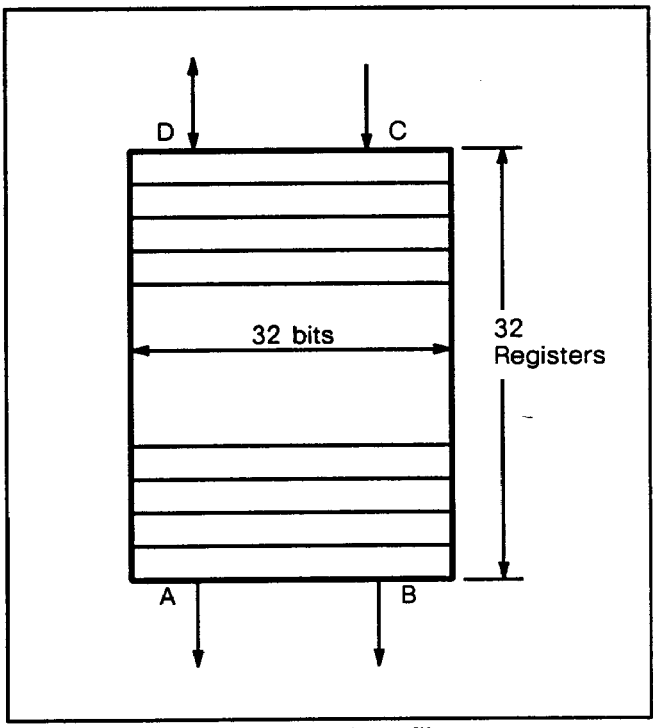


Figure 79. The four-port register file

READ/WRITE CONTROL

Two other fields in the instruction word affect the operation of the register file.

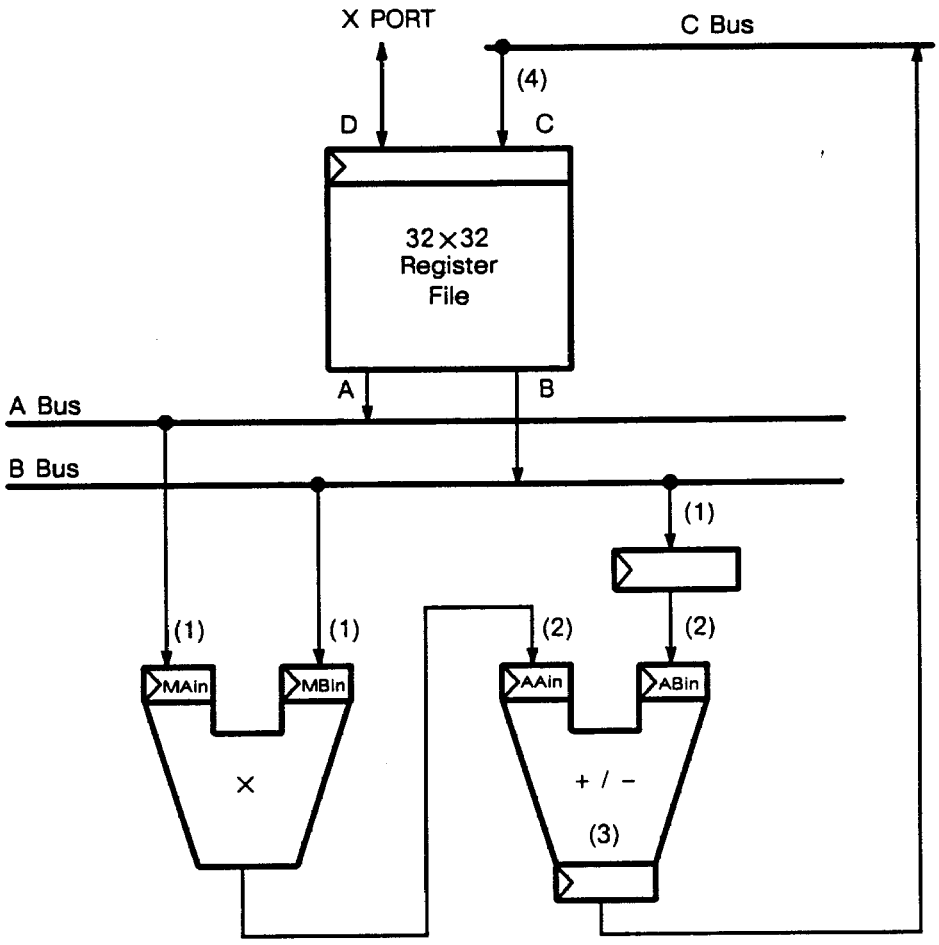
1. The Cwen– bit controls writing of results into the C port. When it is active (low), the result data is written on the fourth cycle of the operation. When writes are disabled, the contents of the register specified by Cadd remain unchanged.

   Register writes may be disabled either to direct a result to a Temporary Register or to allow arithmetic comparisons to modify the Status and Condition Registers without overwriting the contents of a general-purpose register.

2. The IOCt$_{1..0}$ bits control the direction of D port transfers (see page 20 for details). If the C port and the D port attempt to write to the same register file location on the same cycle the register contents are left undefined.

Multiplier/Accumulator

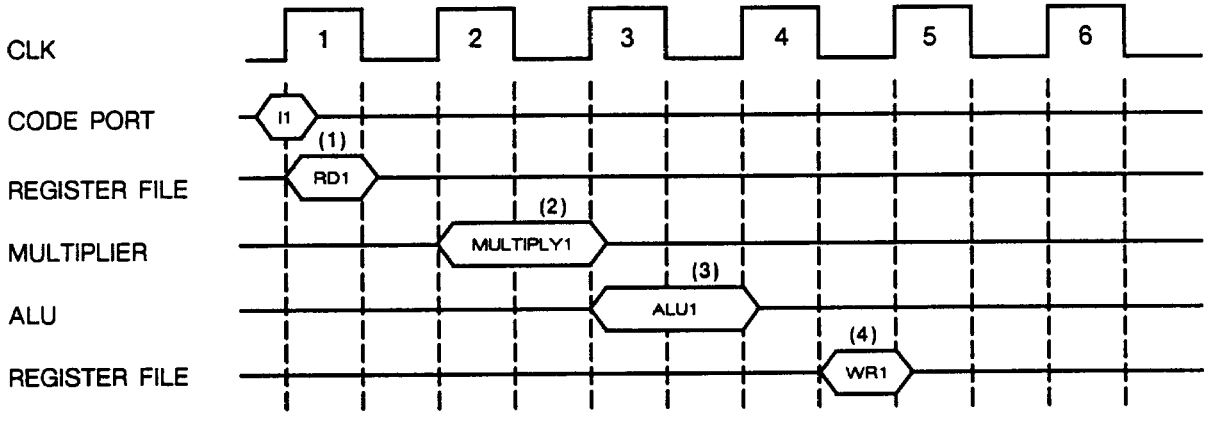
The WTL 3132 and WTL 3332 each have a pipelined multiplier/accumulator. These consist of a floating point multiplier whose output is fed into a floating point ALU (Arithmetic and Logic Unit). All multiplier/accumulator input and output ports can transfer 32-bit data values. Figures 7 and 8 show how operations are pipelined through the multiplier/accumulator.



Note: For clarity, many key features have been omitted from this diagram (see pages 5 and 6 for more detail).

Figure 80. Simple example of multiplier/accumulator timing.

Multiplier/Accumulator, continued



Note: See Figure 7

Figure 81. Timing of MAC operations

MULTIPLIER

The multiplier has two input ports, MAin and MBin. It has one output which can only be connected to the AAin port of the ALU. In the first cycle of an operation, operands are transferred from the register file and fed into MAin and MBin. The multiplication is completed on the second cycle. The intermediate result may be negated before it is passed to the ALU.

ALU

The ALU has two input ports, AAin and ABin. It has one output which is normally connected to the C bus. AAin may be connected to the multiplier's output, so that its result is fed into the ALU. Another operand is fed into the ABin port simultaneously. The ALU completes the function specified by an instruction during its third cycle. The final result is rounded and output to the C bus to be returned to the register file on the fourth cycle.

LATENCY

Because the multiplier/accumulator is pipelined, an operation can be initiated every cycle. The result of an operation is generated three cycles after it is initiated. On the fourth cycle, the result can be returned to the register file or fed straight back into the multiplier/accumulator using the temporary registers or a bypass mode (see pages 13 or 17).

Multiplier/Accumulator, continued

FUNCTION SELECTION

The multiplier/accumulator function is specified by the 3-bit field $F_{2..0}$ in the instruction word as outlined in the function select table (figure 9). A single instruction specifies all of the actions associated with one operation as it passes through the multiplier/accumulator.

When the $F_{2..0}$ field is (0, 0, 0), the operation to be performed is specified by the Badd field according to figure 10.

F_2 F_1 F_0	MNEMONIC	OPERATION	DESCRIPTION
0 0 0	-	Miscellaneous	See figure 10
0 0 1	fsubr	Negate and add	-AAin + ABin
0 1 0	fsub	Subtract	AAin - ABin
0 1 1	fadd	Add	AAin + ABin
1 0 0	-	Reserved	
1 0 1	fmna	Multiply, negate and add	-(MAin × MBin) + ABin
1 1 0	fmns	Multiply, negate and subtract	-(MAin × MBin) - ABin
1 1 1	fmac	Multiply and accumulate	(MAin × MBin) + ABin

Figure 82. Function select field encoding

$Badd_{4..0}$	MNEMONIC	OPERATION	DESCRIPTION
00000	fclsr	Clear Status Register	
00001	fstsr*	Read Status Register	
00010	-	Reserved	
00011	fmode	Load Mode Register	
00100	fabs	Absolute Value	|AAin|
00101	float	Fixed-to-Float	integer → IEEE
00110	fix	Float-to-Fixed	IEEE → integer
00111	flut	Look-up Operation	
01000-11111	-	Reserved	

* fstsr instructions must have their $IOCt_{1..0}$ field set to select a store

Figure 83. Miscellaneous function select encoding

Multiplier/Accumulator, continued

MULTIPLY/ACCUMULATE FUNCTIONS

The WTL 3132/WTL 3332 provide three multiply and accumulate functions: fmac multiplies MAin and MBin and then adds ABin; fmns multiplies MAin and MBin, negates the result and then subtracts ABin; fmna multiplies MAin and MBin, negates the result and then adds ABin.

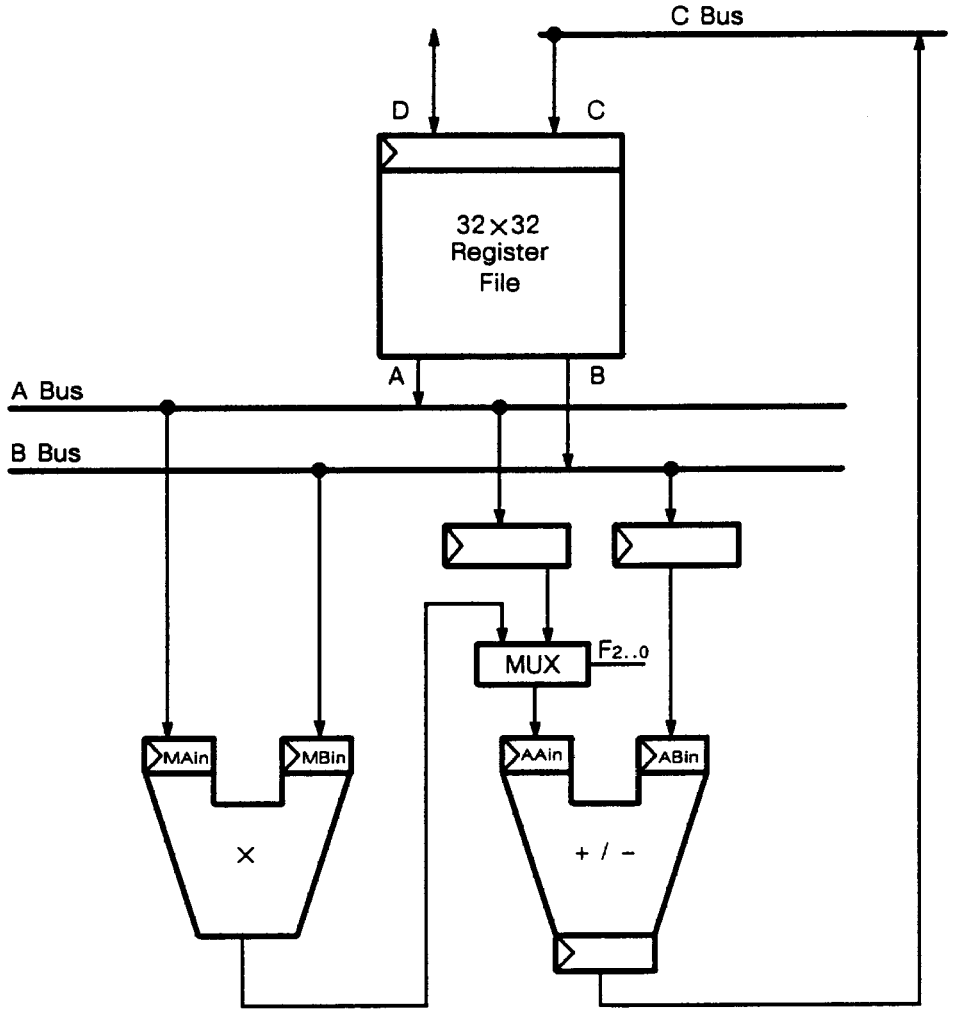
These functions are triadic (they have three input operands). If an IEEE multiply operation is required, the constant 0.0 should be selected as the ABin input.

ALU FUNCTIONS

The WTL 3132/WTL 3332 provide three diadic "ALU only" functions: fadd adds AAin to ABin, fsub subtracts ABin from AAin, fsubr subtracts AAin from ABin.

The $F_{2..0}$ field determines whether the multiplier is bypassed and the ALU's input staged directly into the ALU (see figure 11).

These functions operate in the same number of cycles as the multiply and accumulate functions. This simplifies the programmer's model; every operation has the same latency.



Note: For clarity, many key features have been omitted from this diagram (see pages 5 and 6 for more detail)

Figure 84. "ALU only" operations

MISCELLANEOUS FUNCTIONS

If the function field is equal to zero, then a miscellaneous ALU function will be selected according to the contents of the instruction's Badd field (see figure 10).

8. flut is monadic (that is, it has a single input operand). It takes the value on the A bus as its operand and it returns an approximation to the inverse of this value onto the C bus on its fourth cycle. flut does not attempt to modify the Status, Condition or Zero Registers. It is recommended that the $Abin_{2..0}$ field be set to select the constant 0.0. (See page 38.)

9. fix is a monadic "ALU only" function. It takes a single-precision IEEE format floating point value on the A bus as its operand and returns a 24-bit, sign extended, two's complement integer onto the C bus on its fourth cycle. fix does not attempt to modify the Status or Zero Registers. It is the only instruction that produces an integer result. It is recommended that the $Abin_{2..0}$ field be set to select the constant 0.0. (See page 40.)

10. float is a monadic "ALU only" function. It takes a 24-bit, sign extended, two's complement value on the A bus as its operand and returns a single-precision IEEE format floating point number onto the C bus on its fourth cycle. float does not attempt to modify the Status or Zero Registers. It is the only instruction that requires an integer operand. It is recommended that the $Abin_{2..0}$ field be set to select the constant 0.0. (See page 40.)

11. fabs is a monadic "ALU only" function. It takes the value on the A bus as its operand and returns its absolute value onto the C bus on its fourth cycle. fabs does not attempt to modify the Status Register. The Zero and Condition Registers are modified according to the result unless $Encn_{1..0} = (0,0)$. As with the diadic "ALU only" functions, it will clamp denormalized operands to zero and NaNs to infinity. The $Abin_{2..0}$ field must be set to select the constant 0.0.

12. fmode loads the desired operating modes into the Mode Register (see page 35). Because this operation changes the timing of many operations, the results of the next three operations should be discarded.

    fmode is not cancelled by the NEUT- or ABORT- signals. It should not be executed in a branch shadow.

13. fstsr copies the contents of the Status Register to the X port. It has the same timing as the other fstore operations. (See page 26.) It is recommended that the Cwen- bit be set to prevent register writes, the $Encn_{1..0}$ field to disable updates of the FPCN pin, the $IOCt_{1..0}$ field to an fstore and that the result sent to the register file on its fourth cycle be discarded. fstsr ignores the register address fields.

14. fclsr clears the contents of the Status Register to zero. (See page 26.) It is recommended that the Cwen- bit be set to prevent register writes, the $Encn_{1..0}$ field to disable updates of the FPCN pin, the $IOCt_{1..0}$ to an I/O nop and that the result sent to the register file on its fourth cycle be discarded. fclsr ignores the register address fields.

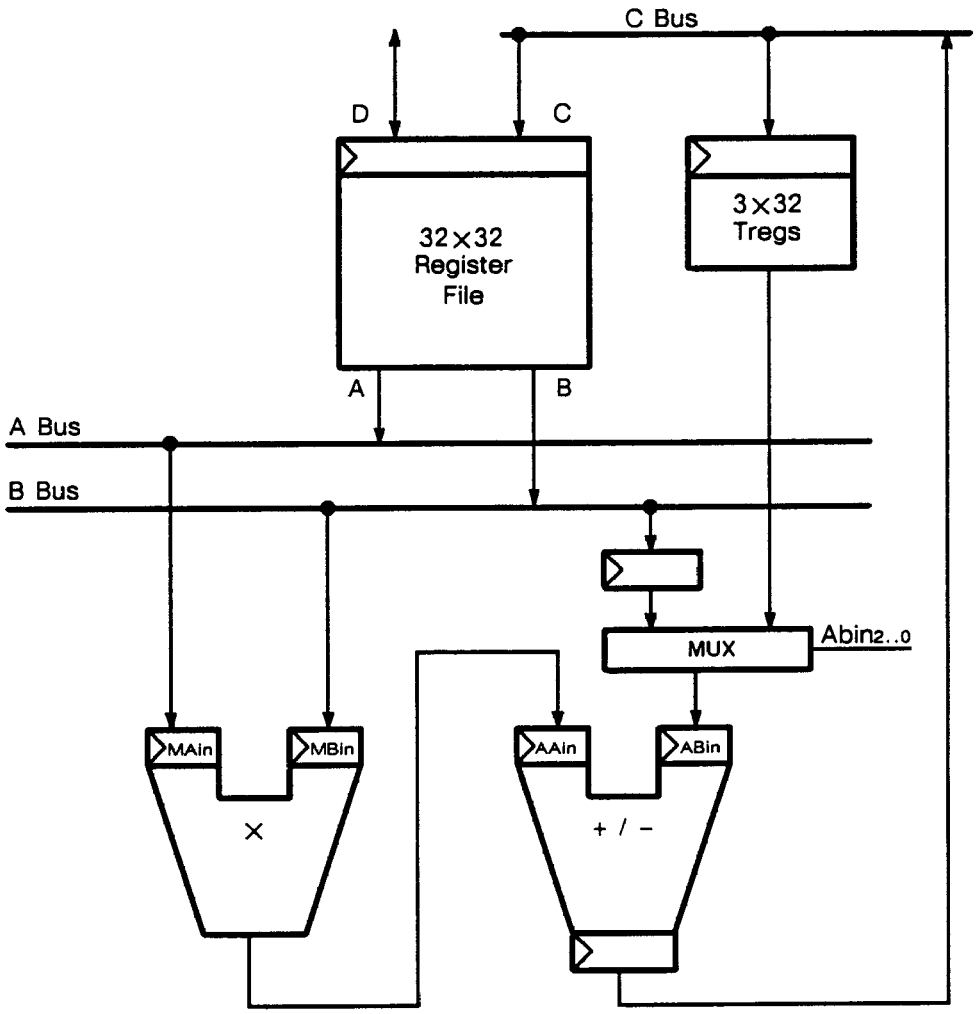
MULTIPLIER/ACCUMULATOR NOP

The WTL 3132/WTL 3332 do not have a dedicated nop instruction. WEITEK software tools use fsub .f0, .f0, .f0 with the Cwen- bit set to disable register writes and the $Encn_{1..0}$ field cleared to disable FPCN updates. This choice of nop causes no state changes.

Temporary Registers

The WTL 3132 and WTL 3332 each include three 32-bit temporary registers (Tregs). They allow values to be recirculated to the ALU without passing through the general-purpose register file. The Tregs are often used as accumulators during successive multiply/accumulate operations. They make it possible to perform a calculation of the form $x = x \pm (y \times z)$ every cycle.

Figures 12 and 13 show how the Tregs are used to feedback operands to the multiplier/accumulator. Figure 15 gives an example of a code sequence that does this.



Note: For clarity, many key features have been omitted from this diagram (see pages 5 and 6 for more detail).

Figure 85. Use of temporary registers

Temporary Registers, continued

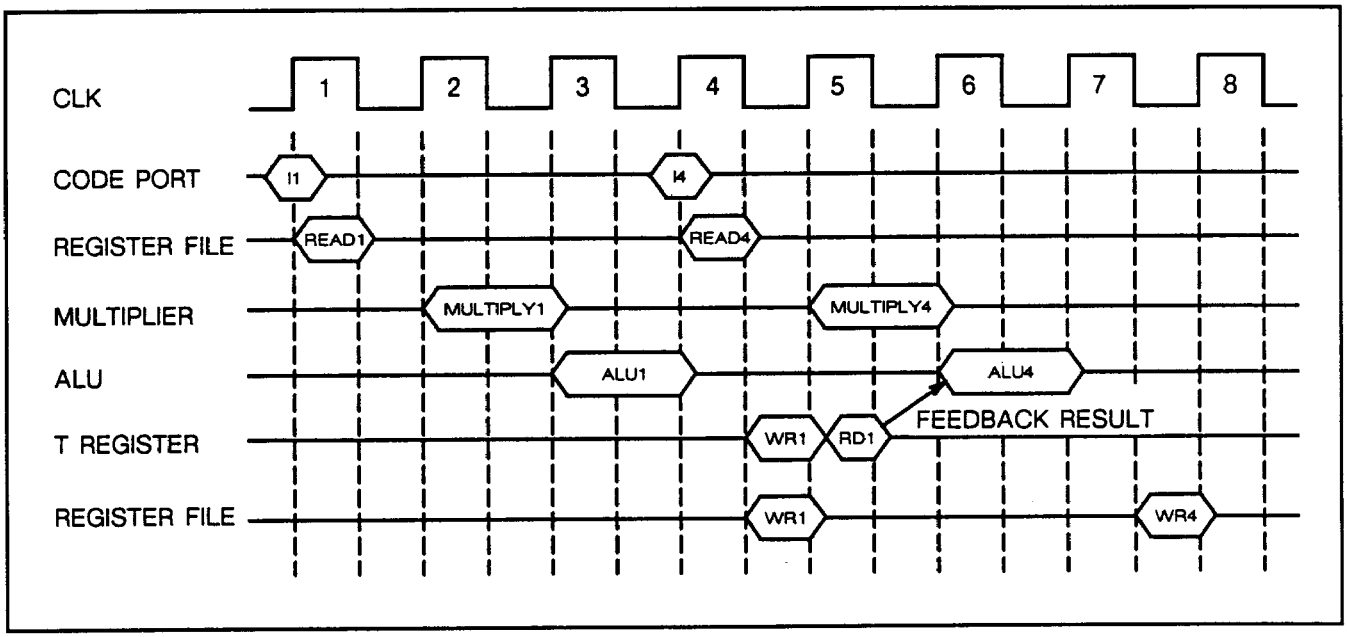


Figure 86. Temporary register timing

WRITING TEMPORARY REGISTERS

The instruction word contains a two-bit field, $Adst_{1..0}$, that determines the destination of the ALU output (see figure 14).

The output of the ALU is always sent to the C bus. If no Treg is selected by the $Adst_{1..0}$ field, then the result is returned only to the register selected by this instruction's Cadd field on its fourth cycle.

If the $Adst_{1..0}$ field selects a Treg in addition to the C bus, it is loaded with the result on the fourth cycle of an operation, just as the Cadd register write occurs. On the next cycle, the contents of the Treg may be input directly to the ABin port. This is illustrated by the example shown in figure 15.

The Cwen– bit of the instruction that writes to the Treg may be held high to prevent the write to the Cadd register. This increases the number of available general-purpose registers.

$Adst_{1..0}$	RESULT DESTINATION
00	$Treg_3$, C bus
01	$Treg_2$, C bus
10	$Treg_1$, C bus
11	C bus

Figure 87. ALU destination select field encoding

Temporary Registers, continued

READING TEMPORARY REGISTERS

The instruction word contains a three-bit field, Abin2..0, that determines the source of the MAC's ABin input.

Three encodings select one of the Tregs (see figure 17). If a Treg is selected, it is copied to the ABin port on the second cycle of the operation.

The Treg may be be read on the cycle after it was written. In figure 15, for example, .t1 gets read during the second cycle of op #4. This is the fifth cycle of op #1.

```
.
.
.
op #1   fmac .f0,  .f1,   0,   .t1
op #2   fmac .f3,  .f4,  .t1,  .f5   —old .t1
op #3   fmac .f6,  .f7,  .t1,  .f8   —Illegal
op #4   fmac .f9,  .f10, .t1,  .f11  —new .t1
.
.
.
```

Notes: A Treg cannot be both written and read in the same cycle. Op #3 *must never* attempt to read the Treg written by op #1.

To make this code interruptable, op #2 and op #3 should not specify .t1 as an operand.

Full details of this syntax are given on page 32.

Figure 88. Use of temporary registers

Internal Data Routing

INTERNAL BUSES

The three main internal buses, A, B and C, move data between the major functional units of the WTL 3132 and WTL 3332. Each of these buses is 32 bits wide and can carry one word per cycle.

The A bus usually carries operands from the register file to the multiplier/accumulator or the divide look-up table. It may also be fed with operands directly from the X port if the Input Bypass Mode is enabled (see page 21).

The B bus usually carries operands from the register file to the multiplier/accumulator. It may also be fed directly with operands from the Y port on the WTL 3332 if selected by the Mbs- bit (see page 23).

The C bus usually carries multiplier/accumulator or flut results back to the general-purpose register file. If the Output Bypass Mode is enabled, then it may be used to feed the results directly to the X port.

If the C bus is not needed to carry the results back to the registers (that is, when they are output directly to the Z port, discarded or fed to a Treg), then its usual direction of transfer may be reversed. It is then used to carry inputs from the X port directly to the multiplier/accumulator. This is only possible by the use of the floadrc operation or the Double-Pump Mode (see page 21).

MULTIPLIER/ACCUMULATOR INPUT PORTS

The multiplier/accumulator has four input ports, MAin, MBin, AAin and ABin. These ports can each receive a 32-bit word per cycle. They all have multiplexers which may be connected to various input sources. The possible selections for each port are described below:

1. MAin usually obtains input from the A bus. Results may be fed from the C bus to the A bus using the Internal Bypass Mode and then into MAin.

2. MBin usually obtains its input from the B bus. Results may be fed from the C bus to the B bus using the Internal Bypass Mode and then into MBin. Y port inputs may be enabled onto the B bus and then into the MBin.

   Alternatively, the C bus can be reversed so that it is carrying inputs from the X port to the MBin port. This prevents the C bus from being used to return results to the register file and is done in conjunction with the floadrc operation or Double-Pump Mode. MBin must select the C bus (see figure 16).

3. AAin usually obtains its input from the A bus. Results may be fed from the C bus to the A bus using the Internal Bypass Mode and then into AAin.

4. ABin usually obtains its input from the B bus. Results may be fed from the C bus to the B bus using the Internal Bypass Mode and then into ABin. Y port inputs may be enabled onto the B bus and then into the ABin.

   The 3-bit Abin2..0 field in the instruction word selects between input from the B bus (as above), input of the constants 0.0 or 2.0, input from one of the Tregs, or input from the C bus (as below) according to figure 17.

   Alternatively, the C bus can be reversed so that it is carrying inputs from the X port to the ABin port. This prevents the C bus from being used to return results to the register file and is done in conjunction with either the floadrc operation or Double-Pump Mode. ABin must select the C bus. When this pathway is used, the ABin data is not delayed: an external register may be needed to synchronize the X and Y inputs.

NOTE: While the C bus is normally used *either* for returning the results to the register file *or* transferring X port inputs directly to a multiplier/accumulator input port, there is one circumstance when it can assume a dual role. It is possible, on the WTL 3332 to feed a value from the Y input port to the ABin port via the B bus and to return a result to the register file and MBin port in the C bus simultaneously.

Mbin-	INPUT
0	B bus
1	C bus

Figure 89. Multiplier input port select field encoding

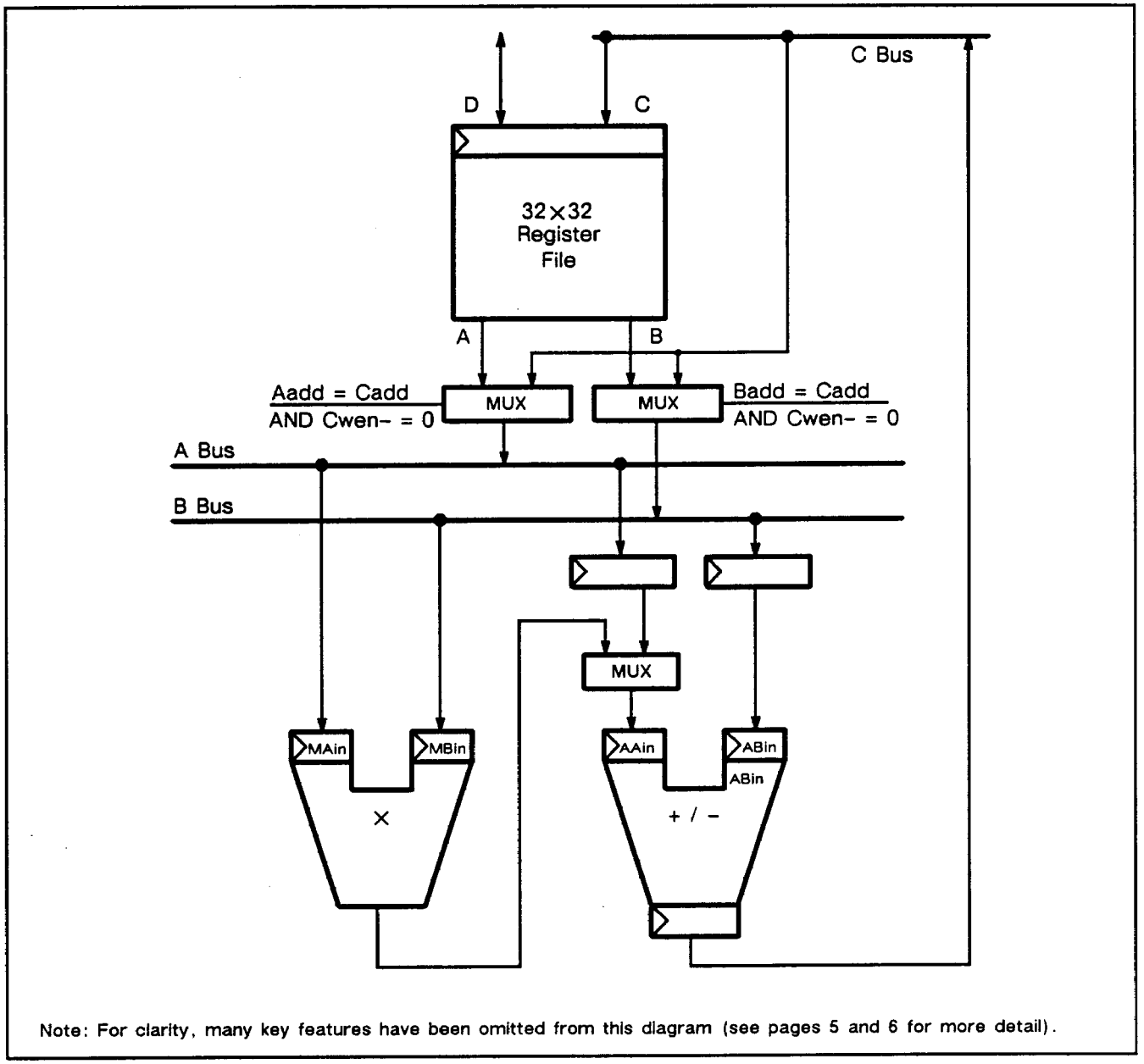
$Abin_{2..0}$	INPUT
000	C bus
001	B bus
010	$Treg_2$
011	$Treg_1$
100	$Treg_3$
101	Reserved
110	2.0
111	0.0

Figure 90. ALU input select field encoding

Internal Data Routing, continued

If the function code specifies an operation that uses the multiplier, it directs data to the MAin or MBin ports. If the multiplier is not used, (that is, in "ALU only" operations), then the data is sent to the AAin or ABin ports automatically. The WTL 3132/WTL 3332 are designed to maintain a consistent latency regardless of the type of operation.

INTERNAL BYPASS MODE



Note: For clarity, many key features have been omitted from this diagram (see pages 5 and 6 for more detail).

Figure 91. Internal bypass routes

Internal Data Routing, continued

A code sequence may often specify the result of one operation to be the operand of a subsequent operation. The WTL 3132/WTL 3332 provide several methods of achieving this:

1. Internal Bypass Mode disabled

   The default mode (M_0 = 0 and M_{11} = 0) of operation is to send the result back to one of the general-purpose registers and then read the new value of this register as the operand.

   Figure 19 shows this code sequence: op #1 specifies .f3 as its destination and op #2–5 all specify .12 f3 as one of their operands. Because the result of op #1 gets written back to .f3 on its fourth cycle, op #5 is the first of the succeeding operations to read the new contents of .f3 as desired. This represents a register-to-register latency of four cycles.

```
.
.
.
op #1   fadd  .f0,  .f1,  .f2
op #2   fadd  .f2,  .f3,  .f4   —old .f2
op #3   fadd  .f2,  .f5,  .f6   —old .f2
op #4   fadd  .f2,  .f7,  .f8   —old .f2
op #5   fadd  .f2,  .f9,  .f10  —new.f2
.
.
.
```

Notes: To make this code interruptable, op #2, op #3 and op #4 should not specify .f2 as an operand.

Full details of this syntax are given on page 32.

Figure 92. No internal bypassing

2. Internal Bypass Mode enabled

   If the Internal Bypass Mode is enabled (M_0 = 1 and M_{11} = 1) then the register-to-register latency is reduced to just three cycles. Figure 18 shows the two internal bypass multiplexers that allow results on the C bus to be copied over to the A or B buses without first being returned to the register file.

   Figure 20 shows a code sequence that uses the bypass mode: op #1 specifies .f2 as its destination and op #2–4 all specify .f2 as one of their operands. In contrast to the previous example, op #4 is fed the result of op #1 on the same cycle that the result is written to the register file.

   The multiplexers operate by comparing the Aadd and Badd address fields to the Cadd address field as they are presented to the register file on each cycle. If Aadd = Cadd, then the value on the C bus is copied to the A bus; and if Badd = Cadd, then the value on the C bus is copied to the B bus. Enough time remains for that value to be latched into a multiplier/accumulator input port before the end of the cycle. In the example, the Cadd field of op #1 matches the Aadd field of op #4 as they are compared on the fourth cycle of op #1, the bypass from C to A buses is opened and op #4 can proceed immediately with the new data. During the same cycle, the result is copied into the Cadd register as usual so that the register file remains consistent with the data values in use.

   Setting mode bit M_0 = 1 enables the C-to-A bus bypass and setting mode bit M_{11} = 1 enables the C-to-B bus bypass. If the Cwen– bit of an instruction is set to prevent register writes, then the Internal Bypass Mode is temporarily suspended on the fourth cycle of that operation; this insures that the register file contents are kept in step with the operands used by each instruction. Similarly, the NEUT–, STALL–, and ABORT– signals cause the Internal Bypass Mode to be suspended as they cancel the register write of an instruction.

```
.
.
.
op #1   fadd  .f0,  .f1,  .f2
op #2   fadd  .f2,  .f3,  .f4   —old .f2
op #3   fadd  .f2,  .f5,  .f6   —old .f2
op #4   fadd  .f2,  .f7,  .f8   —new .f2
.
.
.
```

Notes: To make this code interruptable, op #2 and op #3 should not specify .f2 as an operand.

Full details of this syntax are given on page 32.

Figure 93. Use of internal bypassing

Internal Data Routing, continued

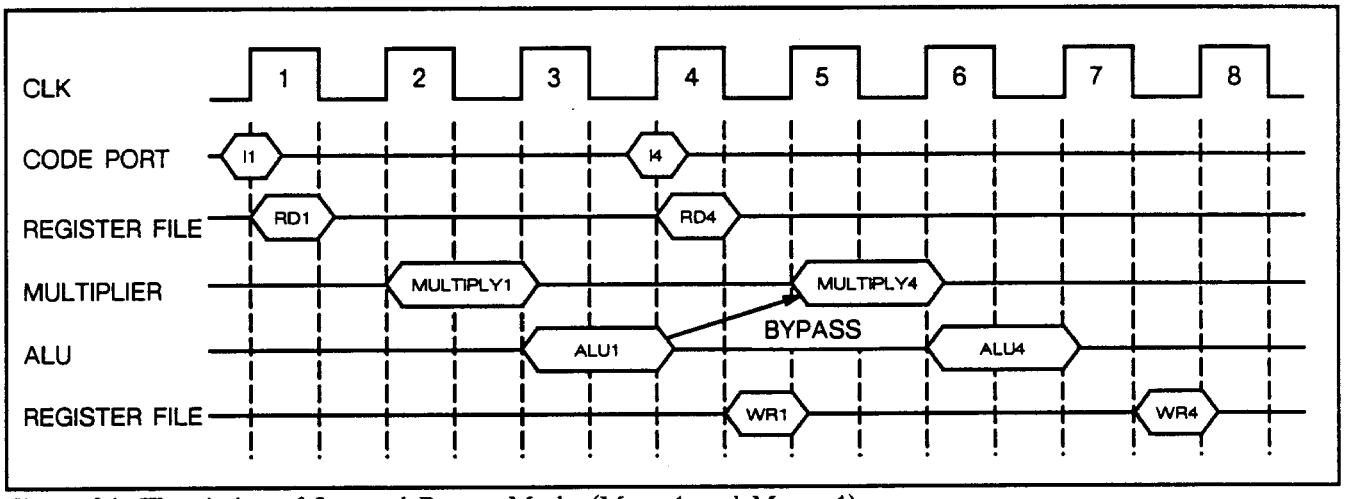


Figure 94. The timing of Internal Bypass Mode (M_0 = 1 and M_{11} = 1)

3. Temporary Registers

   To complete the range of alternative routes available for feeding results back to the multiplier/accumulator, the Treg code example first given in figure 15 is repeated here.

   The temporary register option has the same timing as the Internal Bypass Mode, but only feeds back to the ABin port. The Tregs bring an extra source of operands to the multiplier/accumulator, allowing operations of the form $x = x \pm (y \times z)$ to be executed in a single cycle.

Many of the code examples given read a register after the instruction that modifies it has been initiated. While such code sequences are valid, they are uninterruptable. More detailed coverage of interruptable code may be found on page 33.

```
.
.
.
op #1   fmac .f0,  .f1,  0,   .t1
op #2   fmac .f2,  .f3,  .t1, .f4   —old .t1
op #3   fmac .f5,  .f6,  .t1, .f7   —Illegal
op #4   fmac .f8,  .f9,  .t1, .f10  —new .t1
.
.
.
```

Notes: A Treg cannot be both written and read in the same cycle. Op #3 *must never* attempt to read the Treg written by op #1.

To make this code interruptable, op #2 and op #3 should not specify .t1 as an operand.

Full details of this syntax are given on page 32.

Figure 95. Use of temporary registers

Input/output

The WTL 3132 and WTL 3332 provide different input/output facilities. The WTL 3332 has additional Y and Z ports to increase the bandwidth between the multiplier/accumulator and external components. Sections specific to the WTL 3332 will state this in their headings.

THE X PORT: NORMAL USAGE
(WTL 3132 AND WTL 3332)

All data I/O ports are 32 bits wide. They can all transfer at least one word on each cycle. The memory-to-memory latency can be as low as five cycles (two more than the register-to-register latency). All output buses may be disabled by de-asserting their asynchronous output enable signals.

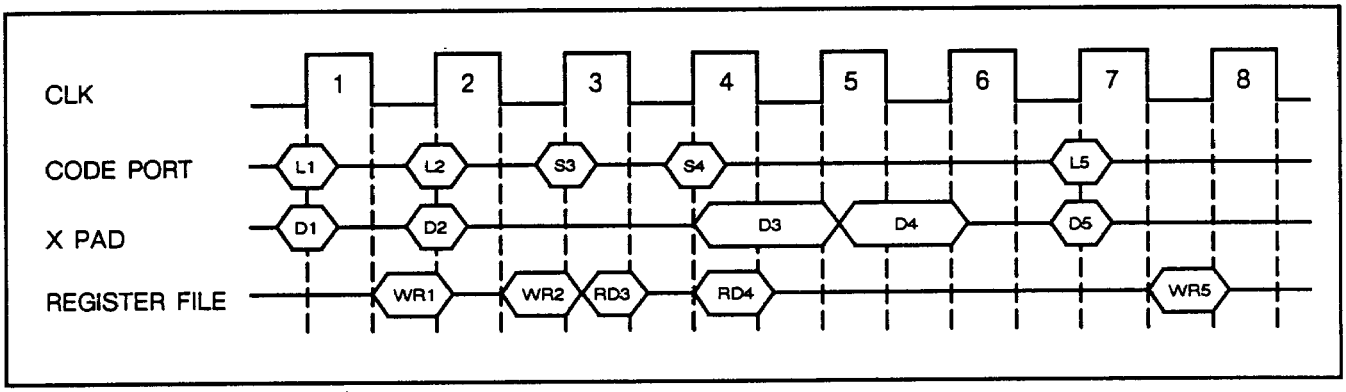


Figure 96. Normal X port I/O timing

The X port normally transfers data to and from the register file D port. The Dadd field selects the register in question and the $IOCt_{1..0}$ field in the instruction word controls the transaction. These I/O transfers always begin during the first cycle of an operation. See figure 24 for the $IOCt_{1..0}$ encoding scheme and figure 23 for normal I/O timing.

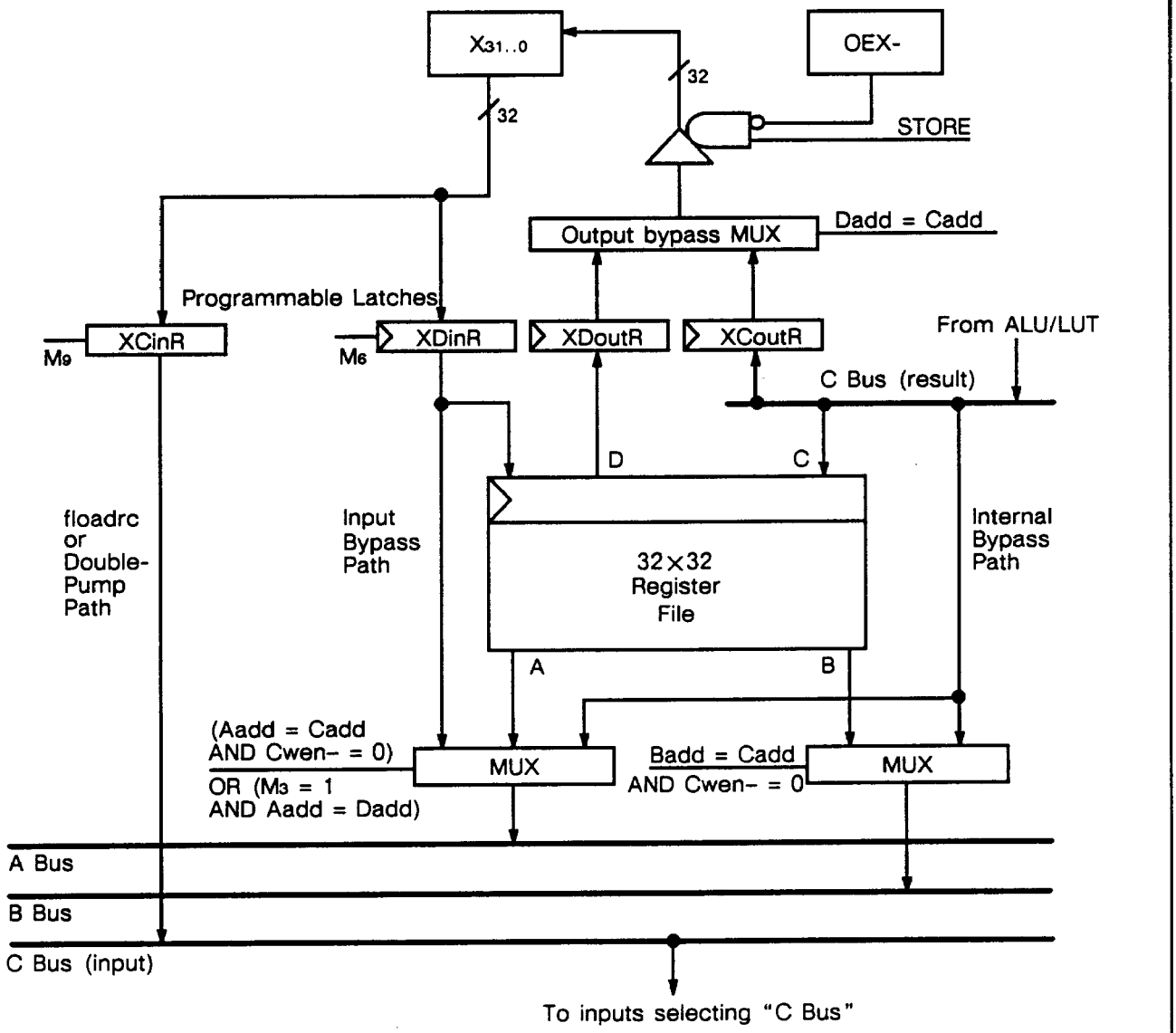
1. The fload operation loads the value at the X port pins into the register selected by Dadd. It is completed by the end of the first cycle. If the register is read on the same cycle, its previous contents will be output.
2. The fstore operation stores the contents of the register selected by Dadd to the X port output register (XDoutR) during the first cycle. This value is driven onto the X port pins on the second cycle. The fstore operation drives the X pads during most of its second cycle and at the start of its third cycle. Input data may not be applied to the pins until partway through its third cycle. The OEX– pin can asynchronously disable the output at any time.
3. The floadrc operation is described on page 22.
4. The I/O nop operation simply disables the X port and ignores any input. It does not prevent the multiplier/accumulator from writing to registers or modifying the state of the condition and exception outputs.

NOTE: An fload should not follow a fstore immediately. At least two I/O nop cycles must be inserted between them if the Coprocessor Load Mode is disabled, and at least one I/O nop if it is enabled (see page 27).

$IOCt_{1..0}$	OPERATION
00	I/O nop
01	floadrc
10	fstore
11	fload

Figure 97. X port I/O control field encoding

Input/output, continued



Note: The C bus can be used either for X port inputs or for results, but not both.

Figure 98. Input/Output Bypass diagram (X port)

INPUT BYPASS MODE (WTL 3132 AND WTL 3332)

During a normal fload operation the X port data must be written to a register one cycle before it can be used as the operand of a subsequent operation. If, however, the Input Bypass Mode is enabled and Aadd = Dadd, then the value at the X port is loaded into both the register selected by Dadd and onto the A bus during the first cycle.

The Input Bypass Mode is enabled when bit M3 of the Mode Register is set to 1 (see page 35).

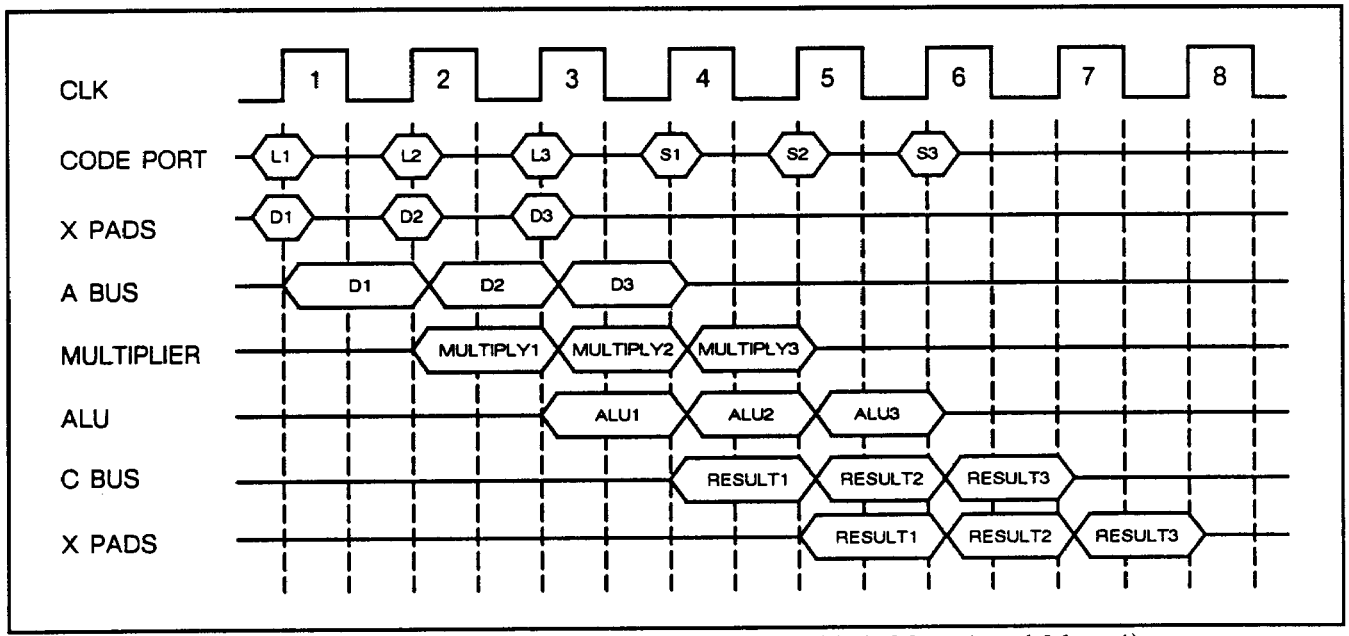


Figure 99. X Port I/O timing (Input and Output Bypass Modes enabled; M_3 = 1 and M_4 = 1)

The MAin port always receives the input data by the end of the first cycle. If the function is "ALU only", the data is staged into the AAin port during the second cycle.

The Input Bypass Mode should not be enabled when the Coprocessor Load Mode is enabled.

FLOADRC OPERATION (WTL 3132 AND WTL 3332)

During a normal fload operation the X port data is copied to the D port, and, if the Input Bypass Mode is activated, the A bus. If the floadrc operation is used instead, the value at the X port is loaded both into the register selected by Dadd and onto the C bus during the first cycle, where it can be selected by one of the multiplier/accumulator input ports, MBin or ABin.

floadrc prevents the multiplier/accumulator from writing the result of a previous instruction to the Cadd register, and the resulting contents of this register are undefined. Results from the divide look-up table preempt the floadrc operation and are written to the Cadd register as usual; floadrc still copies the X port value to the Dadd register. The Cwen- control can be set to prevent unwanted writes to the register specified by the Cadd field of an instruction.

If the MBin multiplexer selects the C bus it will receive the X port data by the end of the first cycle. If the ABin multiplexer selects the C bus it will receive the X port data by the end of the first cycle. Note that the two ports are skewed by one cycle from the programmer's view because the ABin input is not delayed.

floadrc should not be used in an interruptable environment.

OUTPUT BYPASS MODE (WTL 3132 AND WTL 3332)

During a normal fstore operation the multiplier/accumulator result must be written to a register one cycle before it can be output to the X port. If, however, the Output Bypass Mode is enabled and Cadd = Dadd; the multiplier/accumulator result is sent to both the register selected by Cadd and the X port output register (XCoutR) on the same cycle.

The Output Bypass Mode is enabled when bit M_4 of the Mode Register is set (see page 35).

The fstore instruction that specifies the Dadd must start execution on the fourth cycle of the arithmetic instruction that specified the Cadd. The output appears at the X port pins during the second cycle of the fstore instruction. (See figure 26.)

If the Cwen- bit of an instruction is set to prevent register writes, then the Output Bypass Mode is temporarily suspended on the fourth cycle of that operation. This insures that the register file contents are kept in step with the output data.

Input/Output, continued

THE Y PORT: NORMAL USAGE (WTL 3332 ONLY)

The Y port is an input-only port. The Mbs- bit in the instruction word controls whether the B register file port or the Y input port drives the B bus (see figure 28). If the B port is selected, then the data input at the Y port is discarded.

If the MBin port then selects the B bus as input, the multiplier will receive the Y port data by the end of the first cycle. If the ABin port selects the B bus as input, the ALU will receive the Y port data at the end of the next cycle.

Two external data streams may be fed into the multiplier/accumulator through the X and Y ports without being written to the register file. The Y port data can be fed into the MBin or ABin inputs via the B bus. Meanwhile, the X port data can be fed to the MAin or AAin inputs via the A bus using the Input Bypass Mode. Alternatively, the X port data may be fed into the C bus with the floadrc operation.

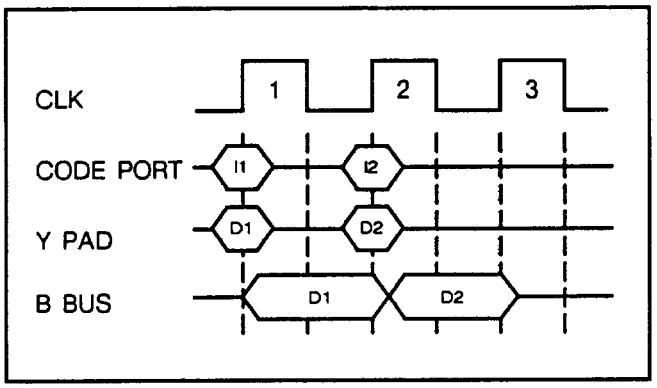


Figure 100. Normal Y port I/O timing

Mbs-	B BUS INPUT SOURCE
0	Register file (B Port)
1	External input (Y Port)

Figure 101. Y port input select field encoding

Y LATE INPUT MODE (WTL 3332 ONLY)

The Y port data may be sampled on the falling edge of CLK instead of its rising edge. This mode is selected by setting bit M_{12} of the Mode Register to 1 (see page 35). It reduces the delay through the YinR register so that the data still arrives at the multiplier/accumulator ports before the end of the first cycle.

The Y port Late Input Mode can be used to demultiplex a high-bandwidth input bus onto the X and Y ports.

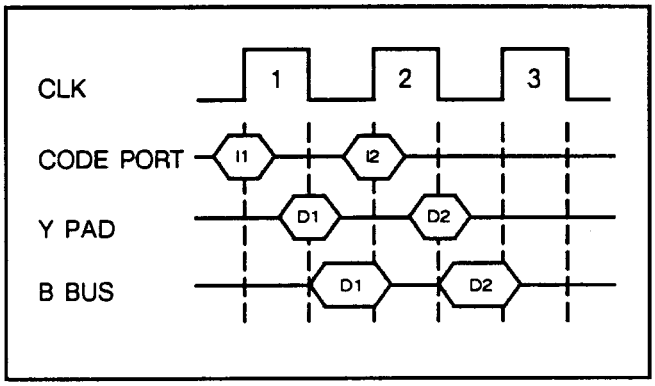


Figure 102. Y port Late Input Mode timing

THE Z PORT: NORMAL USAGE (WTL 3332 ONLY)

The Z port is an output-only port. It outputs the result of a multiply/accumulate operation directly (see figure 30). The result of a flut operation cannot be output to the Z port.

The Z port output register (ZoutR) is loaded on the fourth cycle of an operation; just when the Cadd register should be written. This transfer occurs without regard to any other activity on the C bus.

The Z port output register then drives this value onto the Z port pins. The output occurs during the fifth cycle of the instruction. The Z port output register is updated on every cycle, even if the multiplier/accumulator result is meaningless. The OEZ- pin can asynchronously float the output at any time.

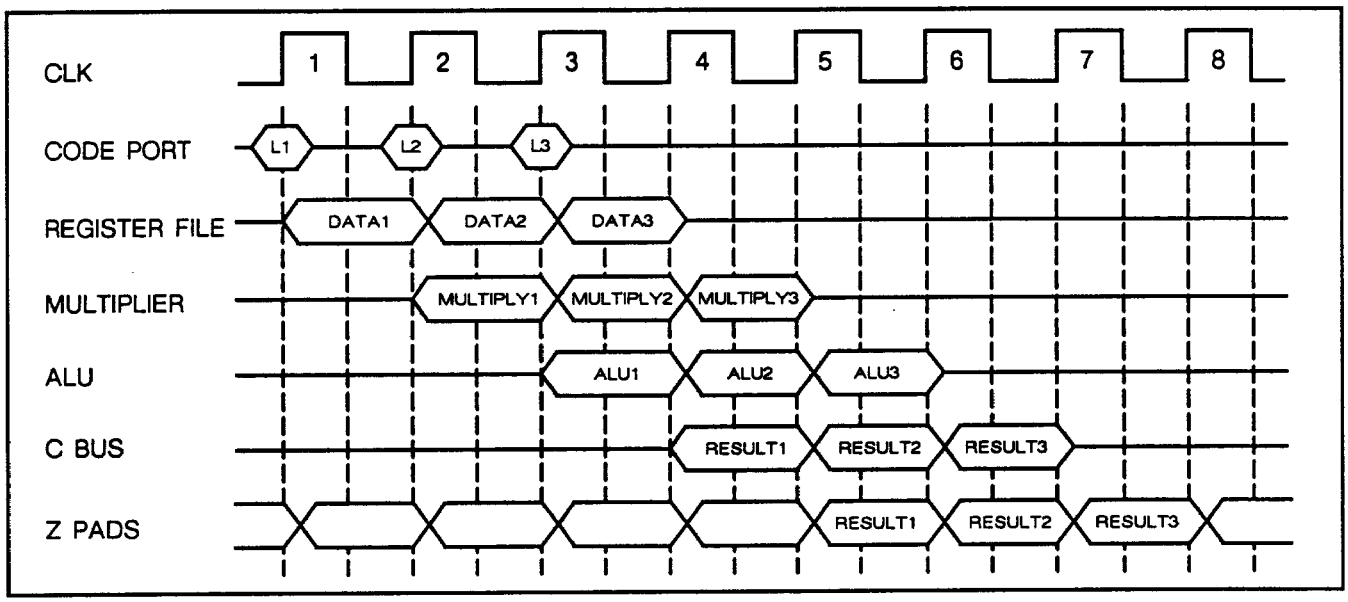


Figure 103. Normal Z port output timing

DOUBLE-PUMP MODE
(WTL 3332 ONLY)

The WTL 3332 multiplier/accumulator has four ports. It can be fed an operand through each of the MAin, MBin and ABin inputs and returns a result via the ALU output on every cycle. Only three external ports are provided, X, Y and Z. Double-Pump Mode allows two inputs to be made via the X port on each cycle; all four multiplier/accumulator ports may then be serviced by the external ports.

Double-Pump Mode is enabled by setting bit M_9 of the Mode Register to 1 (see page 35). If Double-Pump Mode is enabled, the $IOCt_{1..0}$ field in the instruction word must constantly be set to fload.

The first input is latched into the X port on the rising edge of CLK at the beginning of the cycle, as usual. It is written into the Dadd register by the end of the cycle. If the Input Bypass Mode (see above) is enabled and Aadd = Dadd, then this value is also driven onto the A bus and can be latched by the MAin port by the end of the first cycle or the AAin port by the end of the second cycle.

The second input is latched into the X port on the next falling edge of CLK. It is driven onto the C bus and may be latched by the MBin port or the ABin port by the end of the first cycle.

The MBin or ABin port may also receive input data from the Y port via the B bus allowing all three multiplier/accumulator inputs to be fed simultaneously.

The multiplier/accumulator result should be directed to the Z port.

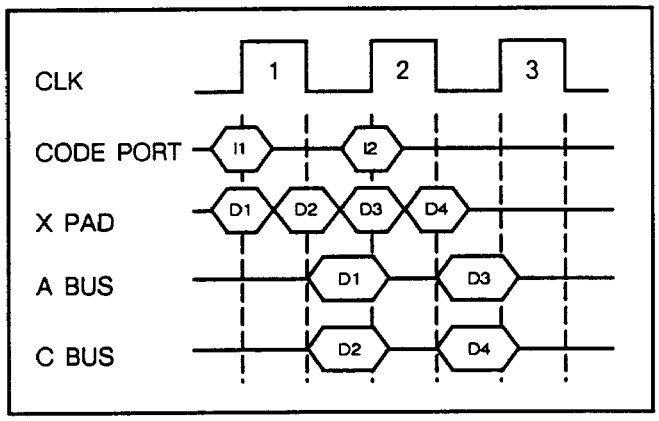


Figure 104. Double-Pump mode timing

System Interfacing

Certain signals on the WTL 3132 and WTL 3332 are provided to communicate control information to and from the other parts of a system.

Two outputs, FPCN and ZERO, indicate the condition of an operation. They can be sent to a sequencer to control instruction branching.

One output, FPEX, signals the occurrence of arithmetic overflow. It can be used to interrupt a host processor to request corrective action.

Three inputs, NEUT–, STALL– and ABORT–, allow the effects of instructions fed into the C port to be canceled. They can be used to make the WTL 3132/ WTL 3332 respond correctly to page faults, interrupts or other system requests.

CONDITION AND ZERO

The WTL 3132/WTL 3332 have a Condition Register and a Zero Register. The multiplier/accumulator attempts to modify the contents of these registers on every cycle.

The instruction word includes a two-bit condition select field, $Encn_{1..0}$, which selectively allows the multiplier/accumulator to succeed in updating the contents of these registers. If both bits are cleared, then the previous state of the registers remains unchanged.

Most functions update the Condition Register according to the sign and magnitude of their result. Miscellaneous functions may set the register for other reasons (see figure 32).

$Encn_{1..0}$ determines the exact condition that will set the Condition Register for each instruction. This allows any of the common comparisons ($>$, $\geq$, $=$, $\leq$, $<$) to be made in one operation. Figure 33 gives the bit encoding.

If the result of an operation is exactly equal to zero and the $Encn_{1..0}$ field is not (0,0); the Zero Register is set to 1. If the result is not zero and $Encn_{1..0}$ is not (0,0); the Zero Register is cleared to 0. If $Encn_{1..0}$ is (0,0); the contents of the Zero Register remain the same.

The contents of the Zero and Condition Registers are copied to the ZERO and FPCN outputs respectively on the fourth cycle of the operation, just as the general-purpose register file write occurs. Bypassing does not affect the timing of these signals. These outputs always drive a logic 0 or 1.

FUNCTION	SET CONDITION REGISTER	SET ZERO REGISTER	SET STATUS REGISTER
fmac	$N \leq 0$, $N < 0$, $N = 0$	$N = 0$	exp (N) ≥ 255
fmns	$N \leq 0$, $N < 0$, $N = 0$	$N = 0$	exp (N) ≥ 255
fmna	$N \leq 0$, $N < 0$, $N = 0$	$N = 0$	exp (N) ≥ 255
fadd	$N \leq 0$, $N < 0$, $N = 0$	$N = 0$	exp (N) ≥ 255
fsub	$N \leq 0$, $N < 0$, $N = 0$	$N = 0$	exp (N) ≥ 255
fsubr	$N \leq 0$, $N < 0$, $N = 0$	$N = 0$	exp (N) ≥ 255
flut	–	–	–
fabs (Note 1)	$N \leq 0$ (only true when $N = 0$)	–	–
fix (Note 2)	$\lvert M \rvert > 2^{22}$	–	–
float (Note 2)	$M < (-2^{23})$ or $M > (2^{23} - 1)$	–	–
fnop	–	–	–

N is the result of an operation; M is the operand.

Notes: 1. fabs only tests for zero when M1 = 0
2. fix and float only test for operand range excess when M1 = 1

Figure 105. Effect of functions on Condition, Zero, and Status Register

Encn1	Encn0	SET ZERO REGISTER	SET CONDITION REGISTER
0	0	-	-
0	1	N=0	$N \leq 0$
1	0	N=0	N<0
1	1	N=0	N=0

Note: Encn1 and Encn0 must be zero for fnop.
Condition Register is set by fix or float when Encn1..0 is (0,1) and the operand exceeds the permitted range.

Figure 106. Encn1..0 encoding

STATUS AND EXCEPTIONS

The WTL 3132/WTL 3332 have a Status Register. If an operation produces a result that is too large to be represented in the IEEE single-precision floating point format the multiplier/accumulator attempts to set the Status Register to 1.

The Mode Register includes an exception control bit, Ms (see page 35). If Ms is set to 1 and an overflow occurs, the Status Register is set to 1. If Ms is cleared to 0, the Status Register is cleared to 0. If Ms is subsequently set to re-enable overflows, the Status Register will contain 0.

The contents of the Status Register are copied to the FPEX output on the fourth cycle of the operation, just as the register file write occurs.

One bit in the Mode Register, Ms, selects the polarity of FPEX. If it is set to 1, then FPEX is active high. If it is cleared to 0, then FPEX is active low. If Ms is cleared, the Status Register is "sticky"; once set it will remain so until an fclsr operation is performed.

Two miscellaneous functions, fstsr and fclsr, allow the Status Register to be read at the X port or for it to be cleared. They take effect during their first cycle. If the fstsr operation is performed, then the IOCt1..0 field must specify a 'store' and its timing is the same as a normal store operation (see figure 34). Only the least-significant bit of the Status Register is guaranteed; the other 31 bits should be masked off when it is read. If the fclsr operation is performed, it is complete by the end of its first cycle.

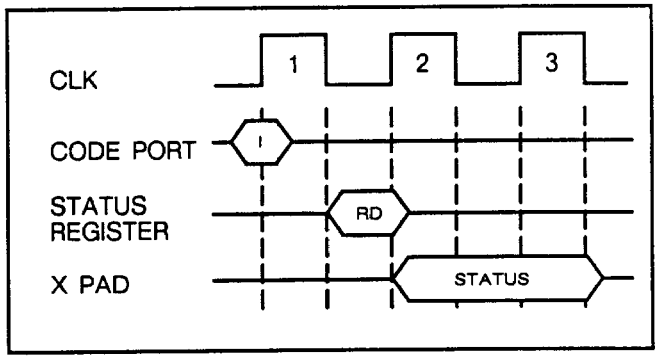


Figure 107. fstsr timing

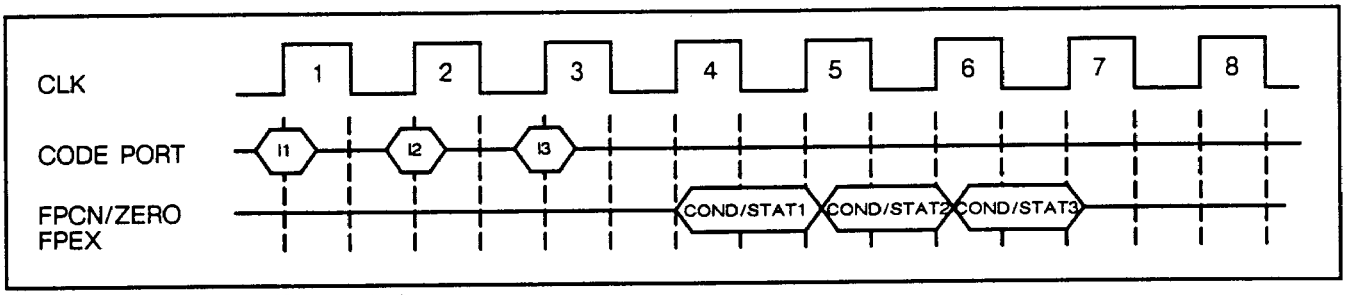


Figure 34a. Condition/Status timing

System Interfacing, continued

COPROCESSOR LOAD MODE

The Coprocessor Load Mode is provided to support systems which generate a data address at the beginning of a cycle and need to latch the data word into the WTL 3132/WTL 3332 later in the same cycle.

If bit M6 of the Mode Register is set, then the data applied to the X port is not sampled until late in the first cycle. Time still remains to write the Dadd register before the end of this cycle. As usual, the next instruction can use this data value as one of its operands.

If this mode is used, neither the Double-pump Mode nor the X port Input Bypass Modes may be enabled.

The number of I/O nops that must be inserted between an fstore and a subsequent fload is reduced from two to one.

The STALL- signal timing is modified internally if the Coprocessor Load Mode is enabled so that it still cancels the fload and fstore operations.

This mode must be selected if the WTL 3132 is to be used in the XL environment (see Appendix A).

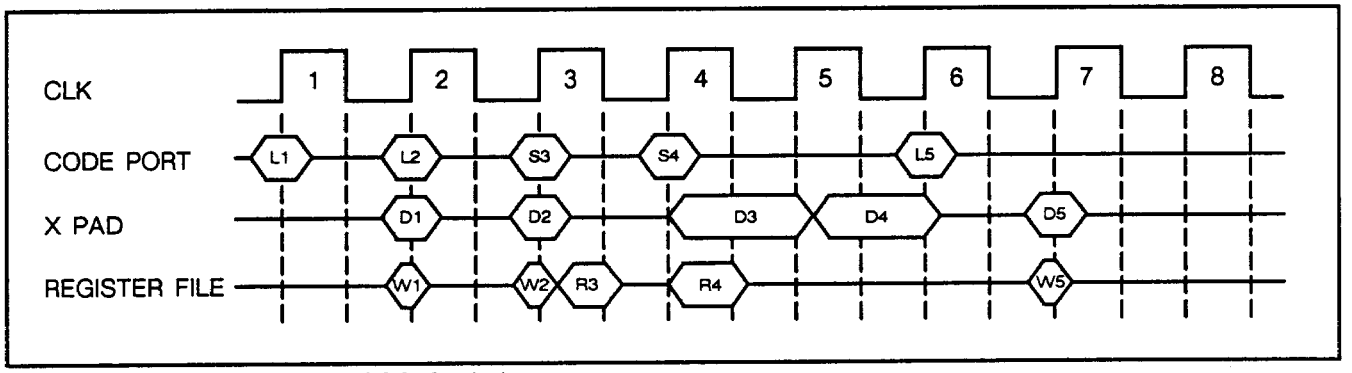


Figure 108. Coprocessor Load Mode timing

NEUT-, STALL- AND ABORT-

These three inputs allow a system to modify the effect of certain instructions dynamically.

1. NEUT-

   Neutralize is used to prevent the execution of instructions in the shadow of a delayed branch operation or during an interrupt service cycle.

   In a system where the sequencer supports delayed branching, it will present the next instruction to the C port as it decides whether to take a branch. If the branch is taken, this instruction must be cancelled before it has any effect on the state of the system. Similarly, if an interrupt occurs, the instruction due to be executed can be cancelled in order to branch to an interrupt service routine. The cancelled instruction is resubmitted for execution on return from interrupt.

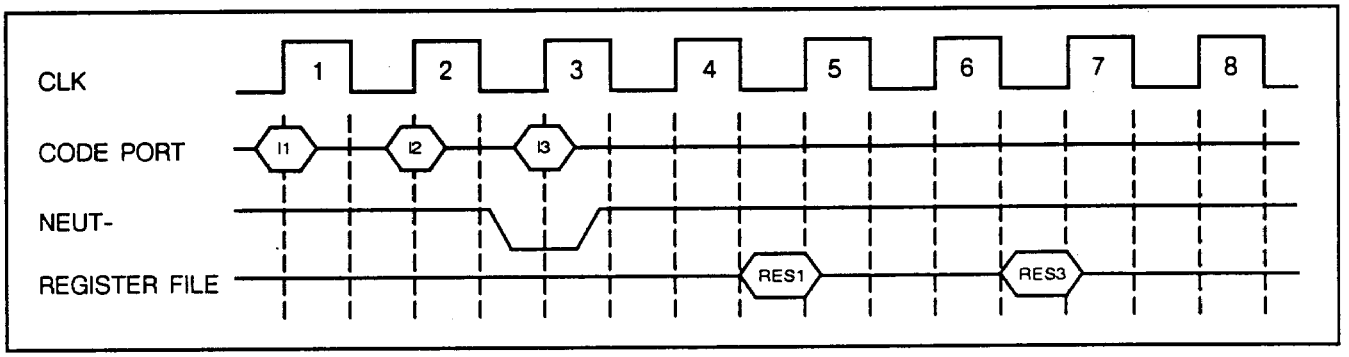


Figure 109. NEUT- timing

The neutralize signal cancels the effect of the current instruction. It prevents the result of this instruction from being written into the register file or temporary registers. It has no effect on fload or fstore operations. This signal is sampled on the rising edge of CLK after the current instruction was fed into the C port.

2. STALL-

STALL- is used to hold off execution until a valid code word is present when the code word is delayed (as in a code memory refresh cycle) or absent (as in a page fault). The next operation can be continually stalled until the correct instruction word is presented to the C port.

The STALL- signal cancels the effect of the next instruction. It prevents the result of this instruction from being written into the register file or temporary registers. It also cancels fload and fstore operations. This signal is sampled at the same time as the next instruction is fed into the C port.

If Coprocessor Load Mode is enabled, the timing of STALL- is modified internally to maintain its usual effect. The fmode and fclsr instructions are also properly stalled.

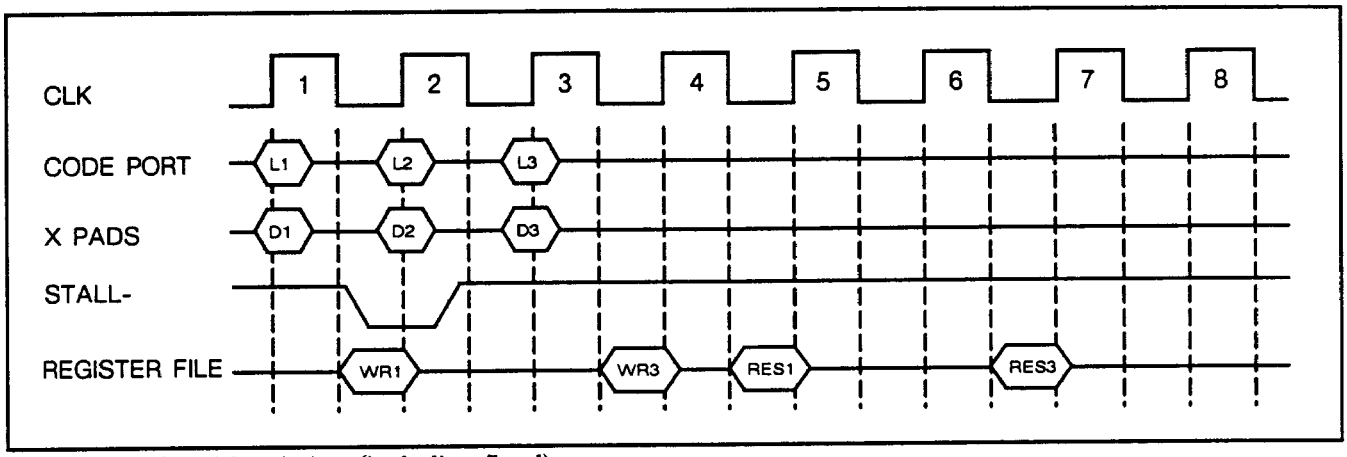


Figure 110. STALL- timing (including fload)

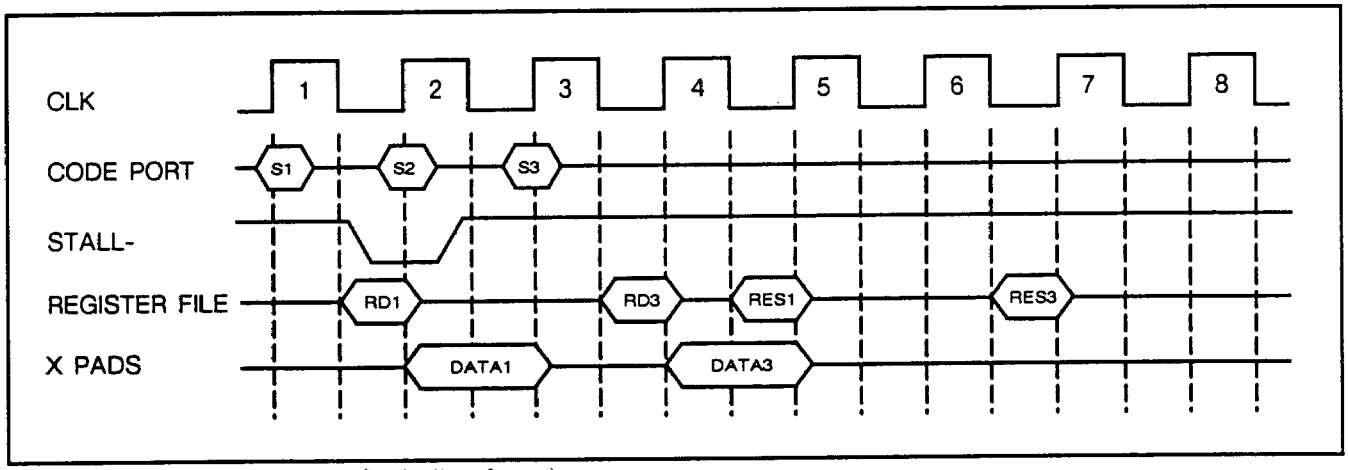


Figure 111. STALL- timing (including fstore)

System Interfacing, continued

3. ABORT-

ABORT- is used to cancel the current and next instructions when the data word is delayed (as in a cache miss) or absent (as in a page fault). If the two cancelled instructions are subsequently resubmitted for execution the processor will behave as if no interrupt occurred.

The ABORT- signal cancels the effect of both the current and next instructions. It prevents the results of these instructions from being written into the register file or temporary registers. It also cancels the I/O operation specified in the next instruction, but not in the current instruction. This signal is sampled at the same time as the next instruction is fed into the C port.

All three of these signals are delayed internally when necessary: for example, the Cadd register write is only disabled on the instruction's fourth cycle. They prevent the cancelled instructions from modifying the Condition or Status Registers, thus preventing the state of the FPCN, FPEX and ZERO outputs from changing.

NOTE: Neither NEUT- nor ABORT- cancel the effect of the fmode operations.

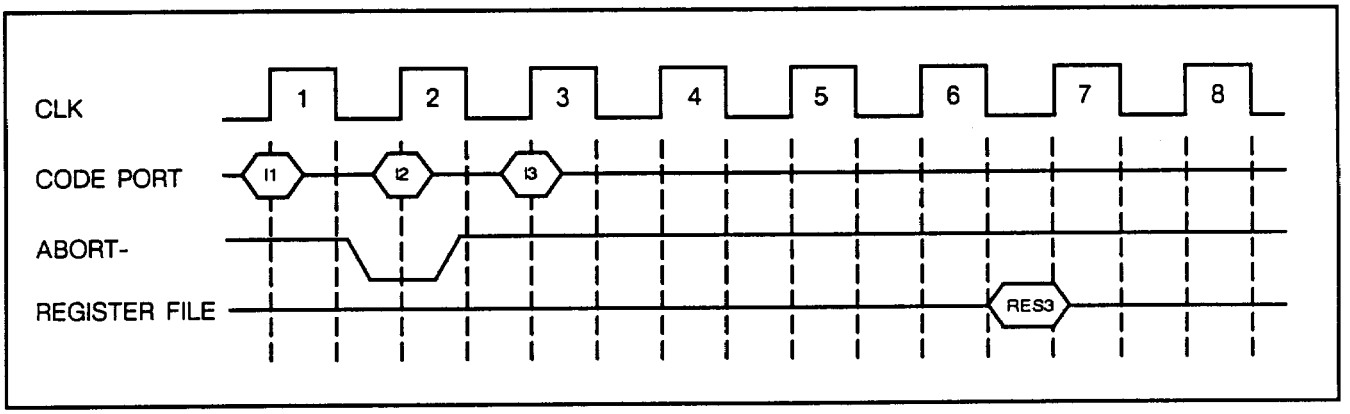


Figure 112. ABORT- timing

Instruction Set

C BIT	FIELD	OPERATION
0	$Encn_0$	Condition Output Select
1	$Encn_1$	
2	Mbin-	MBin Input Select
3	$Adst_0$	ALU Destination Select
4	$Adst_1$	
5	$Abin_0$	ABin Input Select
6	$Abin_1$	
7	$Abin_2$	
8	$Dadd_0$	D Port Register Address
9	$Dadd_1$	
10	$Dadd_2$	
11	$Dadd_3$	
12	$Dadd_4$	
13	$IOCt_0$	I/O Control
14	$IOCt_1$	
15	Cwen-	C Port Write Enable
16	$Cadd_0$	C Port Register Address
17	$Cadd_1$	
18	$Cadd_2$	
19	$Cadd_3$	
20	$Cadd_4$	
21	$Badd_0$	B Port Register Address
22	$Badd_1$	
23	$Badd_2$	
24	$Badd_3$	
25	$Badd_4$	
26	$Aadd_0$	A Port Register Address
27	$Aadd_1$	
28	$Aadd_2$	
29	$Aadd_3$	
30	$Aadd_4$	
31	F_0	Function Code
32	F_1	
33	F_2	
34	Mbs-*	Y Port Input Select*
*Only on WTL 3332		

Figure 113. Instruction format

Instruction Set, continued

FORMAT

The WTL 3132 has a 34-bit instruction word. The WTL 3332 has a 35-bit instruction word because it requires the extra Mbs– bit to control the Y port input. Refer to figure 40 for the location of each field in the instruction word.

1. Mbs– bit. (WTL 3332 only)

   B bus input control bit. May be register file port B or external input port Y.

2. F field.

   Function control ($F_{2..0}$) field. Selects function to be performed on this instruction's operands.

3. Aadd field.

   A register address ($Aadd_{4..0}$) field. Selects location in general-purpose register file to be read out via the A port.

4. Badd field.

   B register address ($Badd_{4..0}$) field. Selects location in general-purpose register file to be read out via the B port. Also encodes miscellaneous functions that require only one operand.

5. Cadd field.

   C register address ($Cadd_{4..0}$) field. Selects location in general-purpose register file to be written into via the C port.

6. Cwen– bit.

   C port write enable bit. Active low.

7. IOCt field.

   I/O control ($IOCt_{1..0}$) field. Selects type of I/O transfer performed via D port.

8. Dadd field.

   D register address ($Dadd_{4..0}$) field. Selects location in general-purpose register file to be read out or written into via the D port.

9. Abin field.

   ALU input multiplexer input control ($Abin_{2..0}$) field. Selects the input source for ALU.

10. Adst field.

    ALU output destination control ($Adst_{1..0}$) field. Selects output destination for ALU. May be the C bus or the C bus and a Temporary Register.

11. Mbin– bit.

    MBin port input control bit. Selects multiplier input source. May be the B bus or C bus.

12. Encn field.

    Condition select ($Encn_{1..0}$) field. Enables a selectable combination of the condition and zero flags onto the FPCN output.

All of the actions specified by these fields are defined in the same instruction word. In this way, all of the stages of an operation, from supplying its operands to storing its results back into a register, are specified together.

Instruction Set, continued

MNEMONICS

The mnemonics shown in figures 41 through 43 are those used to control the XL-3132 in the XL-Series programming environment. They are given here to simplify understanding of the programming model and to provide a syntax in which to present the programming examples.

These mnemonics represent a subset of the functions available on the WTL 3132/WTL 3332. In particular, the Input Bypass Mode is disabled and the Output Bypass, Internal Bypass, and Coprocessor Load Modes are enabled unless otherwise noted. The user is free to enhance or disregard this suggested programming model according to system requirements.

I/O SELECTION	OPERAND	
	Source	Destination
fload	X port	.f0-31
fstore	.f0-31	X port

Note: If no I/O operation is specified, an I/O nop will be selected in the $IOCt_{1..0}$ instruction field.

Figure 114. Recommended mnemonics

OPERAND NOTATION	DESCRIPTION
.f0–31	Thirty-two, 32-bit general purpose registers
0	Constant "0.0"
2	Constant "2.0"
.t1–3	Three temporary registers

Figure 115. Recommended mnemonics

FUNCTION	OPERAND SELECTIONS			
	SOURCE (Aadd)	SOURCE (Badd)	SOURCE (Tregs)	DESTINATION (Cadd)
fmac	.f0-31	.f0-31	0, 2, or .t1–3	.f0-31 and/or .t1-3
fmns	.f0-31	.f0-31	0, 2, or .t1–3	.f0-31 and/or .t1-3
fmna	.f0-31	.f0-31	0, 2, or .t1–3	.f0-31 and/or .t1-3
fadd	.f0-31	.f0-31, 0, 2, or .t1–3		.f0-31 and/or .t1-3
fsub	.f0-31	.f0-31, 0, 2, or .t1–3		.f0-31 and/or .t1-3
fsubr	.f0-31	.f0-31, 0, 2, or .t1–3		.f0-31 and/or .t1-3
flut	.f0-31			.f0-31
fabs	.f0-31			.f0-31 and/or .t1-3
fix	.f0-31			.f0-31 and/or .t1-3
float	.f0-31			.f0-31 and/or .t1-3
fnop	-			

Figure 116. Recommended mnemonics

Instruction Set, continued

CODE CONSTRAINTS

The following set of rules prevents illegal code sequences:

1. All instructions must avoid writing to the Cadd register and Dadd register simultaneously. Thus no fload operation with Dadd = .fx may start on the fourth cycle of an operation with Cadd = .fx.

```
.
.
.
op#1   fadd  .f?, .f?, .fx
op#2   fadd  .f?, .f?, .f?;  fload.fx
op#3   fadd  .f?, .f?, .f?;  fload.fx
op#4   fadd  .f?, .f?, .f?;  fload.fx   —Illegal
op#5   fadd  .f?, .f?, .f?;  fload.fx
.
.
.
```

Figure 117.

2. Because the X port output is driven on the cycle after an fstore operation is specified, an fload cannot follow an fstore immediately. At least one I/O nop must intervene if the Coprocessor Load Mode is enabled (M_6 = 1), or two I/O nops if it is disabled (M_6 = 0) (see page 27).

```
.
.
.
op#1   fadd  .f?, .f?, .f?;  .fstore.f?
op#2   fadd  .f?, .f?, .f?;  .fload.f?  —Illegal
op#3   fadd  .f?, .f?, .f?;  .fload.f?
.
.
.
```

Figure 118. Coprocessor Load Mode enabled, M_6 = 1

```
.
.
.
op#1   fadd  .f?, .f?, .f?;  fstore.f?
op#2   fadd  .f?, .f?, .f?;  fload.f?   —Illegal
op#3   fadd  .f?, .f?, .f?;  fload.f?   —Illegal
op#4   fadd  .f?, .f?, .f?;  fload.f?
.
.
.
```

Figure 119. Coprocessor Load Mode disabled, M_6 = 0

3. No temporary register can be written and read on the same cycle. Thus no operation that selects .tx as an operand register may start on the third cycle of an operation with Cadd = .tx.

```
.
.
.
op#1   fmac  .f?, .f?,  0,  .tx
op#2   fmac  .f?, .f?,  .tx, .f?
op#3   fmac  .f?, .f?,  .tx, .f?   —Illegal
op#4   fmac  .f?, .f?,  .tx, .f?
.
.
.
```

Figure 120.

Instruction Set, continued

If code is to be interruptable and respond correctly to the NEUT–, STALL– and ABORT– signals, then these additional rules must also be followed. They all prevent delayed register writes from modifying operand values in a time-dependent fashion.

4. No operation with Aadd or Badd = .fx may start after the first cycle and before the fourth cycle of an operation with Cadd = .fx. (If the Internal Bypass Mode is disabled ($M_0 = 0$ and $M_{11} = 0$), this becomes the fifth cycle).

```
.
.
.
op#1  fadd .f?, .f?, .fx
op#2  fadd .fx, .f?, .f?   —Illegal
op#3  fadd .fx, .f?, .f?   —Illegal
op#4  fadd .fx, .f?, .f?
.
.
.
```

Figure 121. Internal Bypass Mode enabled, $M_0 = 1$ and $M_{11} = 1$

```
.
.
.
op#1  fadd .f?, .f?, .fx
op#2  fadd .fx, .f?, .f?   —Illegal
op#3  fadd .fx, .f?, .f?   —Illegal
op#4  fadd .fx, .f?, .f?   —Illegal
op#5  fadd .fx, .f?, .f?
.
.
.
```

Figure 122. Internal Bypass Mode disabled, $M_0 = 0$ and $M_{11} = 0$

5. No operation that selects .tx as an operand register may start after the first cycle and before the fourth cycle of an operation with Cadd = .tx.

```
.
.
.
op#1  fmac .f?, .f?, 0,  .tx
op#2  fmac .f?, .f?, .tx, .f?  —Illegal
op#3  fmac .f?, .f?, .tx, .f?  —Illegal
op#4  fmac .f?, .f?, .tx, .f?
.
.
.
```

Figure 123.

6. No operation with Aadd or Badd = .fx may start on the same cycle as an fload where Dadd = .fx.

```
.
.
.
op#1  fadd .fx, .f?, .f?; fload.fx  —Illegal
op#2  fadd .f?, .f?, .f?
.
.
.
```

Figure 124.

7. The NEUT– line does not cancel fload and fstore, so when it is used to cancel the effect of an instruction in the shadow of a delayed branch operation (as in the XL-Series), this instruction should not perform I/O transfers. (This is not necessary when NEUT– is asserted during an interrupt response cycle because the cancelled instruction is resubmitted for execution.)

In the examples, the notation .f? is used to indicate any register except .fx.

Initialization

MODE REGISTER

The Mode Register controls which of the special modes are enabled. Normally, it is initialized to the desired state and is not subsequently altered. Some mode bits are provided to maintain backward compatibility with previous versions of the WTL 3132/WTL 3332. Other bits are reserved and should be set to the value specified in figure 52.

MODE BIT	LOGIC VALUE	DESCRIPTION
M_0	0	Internal Bypass Mode (Aadd = Cadd) disabled
	1	Internal Bypass Mode (Aadd = Cadd) enabled
M_1	0	fix rounds to negative infinity
	1	fix rounds to nearest (enable fix/float range test)
M_2	–	Reserved: should be cleared to 0
M_3	0	Input Bypass Mode disabled
	1	Input Bypass Mode enabled
M_4	0	Output Bypass Mode disabled
	1	Output Bypass Mode enabled
M_5	0	FPEX Output disabled
	1	FPEX Output enabled
M_6	0	Co-processor Load Mode disabled
	1	Co-processor Load Mode enabled
M_7	–	Reserved: should be set to 1
M_8	0	FPEX active low and "sticky"
	1	FPEX active high
M_9	0	Double-pump Mode disabled
	1	Double-pump Mode enabled
M_{10}	–	Reserved: should be set to 1
M_{11}	0	Internal Bypass Mode (Badd = Cadd) disabled
	1	Internal Bypass Mode (Badd = Cadd) enabled
M_{12}	0	Y Late Input Mode disabled
	1	Y Late Input Mode enabled

Figure 125. Mode selection table

Initialization, continued

The Mode Register is loaded by the fmode operation. This causes the Aadd, Cadd and ABin2..0 fields in the instruction word to be loaded into the Mode Register as shown by figure 53. fmode completes by the end of its first cycle.

C BIT	NORMAL USE	MODE BIT	COMMENT
0	$Encn_0$	0	FPCN is disabled during fmode
1	$Encn_1$	0	
2	Mbin-	0	
3	$Adst_0$	1	ALU destination is C bus only
4	$Adst_1$	1	
5	$Abin_0$	M_{10}	
6	$Abin_1$	M_{11}	
7	$Abin_2$	M_{12}	
8	$Dadd_0$	0	
9	$Dadd_1$	0	
10	$Dadd_2$	0	
11	$Dadd_3$	0	
12	$Dadd_4$	0	
13	$IOCt_0$	0	I/O nop specified
14	$IOCt_1$	0	
15	Cwen-	1	C port register writes disabled
16	$Cadd_0$	M_5	
17	$Cadd_1$	M_6	
18	$Cadd_2$	M_7	
19	$Cadd_3$	M_8	
20	$Cadd_4$	M_9	
21	$Badd_0$	1	fmode function code
22	$Badd_1$	1	
23	$Badd_2$	0	
24	$Badd_3$	0	
25	$Badd_4$	0	
26	$Aadd_0$	M_0	
27	$Aadd_1$	M_1	
28	$Aadd_2$	M_2	
29	$Aadd_3$	M_3	
30	$Aadd_4$	M_4	
31	F_0	0	Miscellaneous function selector
32	F_1	0	
33	F_2	0	
34	Mbs-*	0	

*Only on WTL 3332

Figure 126. Load Mode Register instruction format

Initialization, continued

Changing the contents of any bits in the Mode Register will have undefined effects on any currently executing instructions including I/O operations. The state of all registers should be initialized after execution of the fmode instruction.

Some combinations of modes are not allowed. These are detailed in figure 54.

MODE #		(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Internal Bypass Mode (M_0 = 1 and/or M_{11} = 1)	(1)	–	✓	✓	✓	x	✓	x	✓
Input Bypass Mode (M_3 = 1)	(2)	✓	–	✓	✓	✓	x	✓	✓
Output Bypass Mode (M_4 = 1)	(3)	✓	✓	–	✓	x	✓	✓	✓
Y Late Input Mode (M_{12} = 1)	(4)	✓	✓	✓	–	✓	✓	✓	✓
Double-Pump Mode (M_9 = 1)	(5)	x	✓	x	✓	–	x	✓	x
Coprocessor Load Mode (M_6 = 1)	(6)	✓	x	✓	✓	x	–	x	✓
Use of floadrc Operation	(7)	x	✓	✓	✓	✓	x	–	x
Use of NEUT–, STALL–, or ABORT– Inputs	(8)	✓	✓	✓	✓	x	✓	x	–

x = Should not be enabled together, ✓ = Can be enabled together

Note: Although (7) and (8) are not selected by mode bits, the user is able to avoid the use of such functions or control lines
Double-Pump Mode (5) requires floadrc (7) to be used every cycle

Figure 127. Mode exclusion table

RESET SEQUENCE

Before initializing the contents of the Mode Register, the WTL 3132/WTL 3332 must be set to a stable state after power up.

Repeating nop and I/O nop instructions for at least four cycles will flush the multiplier/accumulator pipeline and allow the internal states to become well-defined. Until this sequence terminates, the rest of the system should ignore the contents of the data buses and the state of the ZERO, FPCN and FPEX pins.

The registers should then all be initialized to known values (including the Mode, Condition, Status and Tregs) while nops continue to be input. The WTL 3132/WTL 3332 is then able to begin normal operation.

Division

DIVIDE LOGIC UNIT

The WTL 3132/WTL 3332 have a divide logic unit. This unit consists of a look-up table ROM and three delay stages. The first cycle of the flut operation transfers the Aadd operand (*a*) to the divide logic unit. During the next two cycles this operand selects a seed value for the reciprocal of the operand (1/*a*) from the look-up table. This result is written to the Cadd register on the fourth cycle.

If the Internal Bypass Mode is enabled, the result can be copied to a multiplier/accumulator input port at the same time that the Cadd register is being written.

The look-up result is an IEEE single-precision number whose fraction is accurate to seven bits of precision. If the input is positive or negative infinity (greater than #7F800000 or less than #FF800000), the result is zero. If the input is zero, the result is #7FFFFFFF (which gets clamped to #7F800000 during refinement).

flut does not update the Zero, Condition, or Status Registers.

NOTATION:

a = divisor (.f1)

R_0 = seed for $1/a$ (.f31)

R_1 = first approximation (.f31)

R_2 = second approximation (.f30)

b = dividend (.f0)

b/a = result (.f0)

ALGORITHM:

$R_1 = R_0 \times (2 - a \times R_0)$

$R_2 = R_1 \times (2 - a \times R_1)$

Cycle #	Opcode					I/O	Comment
1	flut	.f1,			.f31		R_0 ($\simeq 1/a$)
2	fnop						
3	fnop						
4	fmna	.f1,	.f31,	2,	.f30		$2 - a \times R_0$
5	fnop						
6	fnop						
7	fmac	.f31,	.f30,	0,	.f31		$R_1 = R_0 \times (2 - a \times R_0)$
8	fnop						
9	fnop						
10	fmna	.f1,	.f31,	2,	.f30		$2 - a \times R_1$
11	fnop						
12	fnop						
13	fmac	.f31,	.f30,	0,	.f30		$R_2 = R_1 \times (2 - a \times R_1)$
14	fnop						
15	fnop						
16	fmac	.f0,	.f30,	0,	.f0		$B \times 1/a$
17	fnop						
18	fnop						
19	fnop					; fstore .f0	store b/a
20	fnop						b/a on X port

Notes: Internal bypass enabled (M_0 = 1 and M_{11} = 1)
X port output bypass enabled (M_4 = 1)

Figure 128. Recommended division sequence

Division, continued

DIVIDE CODE SUPPORT

The initial approximation to $1/a$ has to be refined by successive approximation. This accurate value of $1/a$ must then be multiplied by b in order to complete the divide operation (b/a).

The programmer or compiler has to supply code to support the following sequence of operations:

1. Execute flut to obtain seed value.
2. Iterate from this value to obtain an accurate divisor inverse using the *Newton-Raphson* algorithm.
3. Multiply the dividend by the inverse of the divisor just generated.

For division, the *Newton–Raphson* algorithm converges quadratically. Theoretically, the number of bits of precision doubles with each iteration. Thus, two iterations should provide the full 23 bits of precision representable by the IEEE 32-bit format. Quantization errors introduced by rounding, however, can prevent the lsb from being accurate. The code example provides b/a to around 22 bits of precision.

Data Format

32-BIT FLOATING POINT (IEEE STANDARD)

The IEEE standard 32-bit floating point word divides into three fields: a sign bit, an 8-bit exponent and a 23-bit fraction field (shown in figure 56).

The value contained in the 8-bit exponent field ranges from −127 to 128 (#00 to #FF) (shown in figure 57). The fraction is multiplied by two raised to this power to produce a floating point value.

The significand field contains the 23-bit fraction and the hidden bit. Inserted during arithmetic processing, the hidden bit has a value of one for all normalized numbers and zero for zero. The fraction is the 23 bits to the right of the hidden bit. Bit F_{22} has a value of 2^{-1}; bit F_0 has a value of 2^{-22}; the hidden bit has a value of 2^0.

All constants are in this IEEE format.

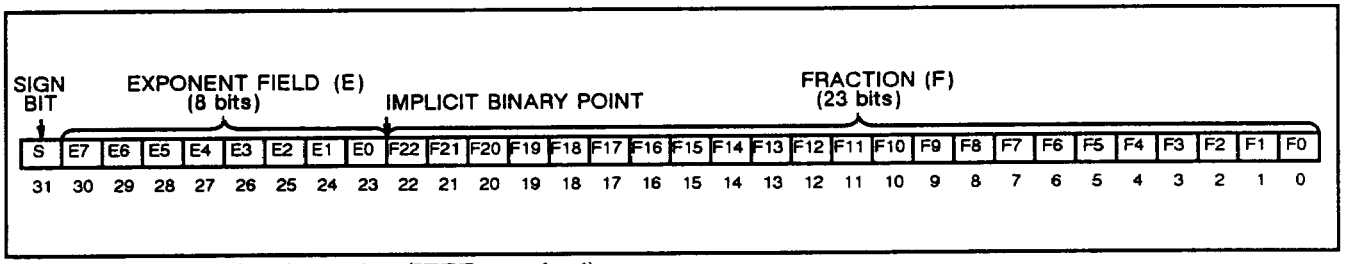


Figure 129. 32-bit floating point (IEEE standard)

The value of an IEEE floating point number is determined by the following:

E	F	VALUE	DESCRIPTION
1-254	Any	$(-1)^S$ (1.F) 2^{E-127}	Normalized number (NRN)
0	Any	$(-1)^S$ 0.0	Zero

Figure 130. 32-bit floating point value

Data Format, continued

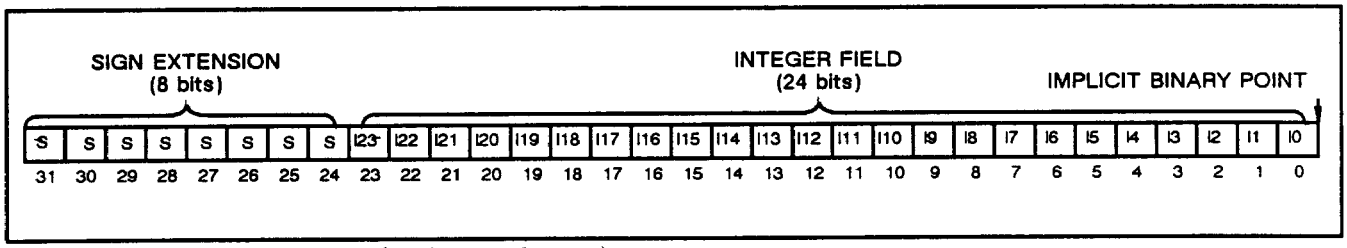


Figure 131. 24-bit fixed point (two's complement)

24-BIT TWO'S COMPLEMENT INTEGERS

The value of the 24-bit integer field shown in figure 58 can range from (-2^{23}) to $(2^{23}-1)$ and must be sign-extended to 32-bits to be compatible with the WTL 3132/WTL 3332 format. The eight-bit sign extension field is a repeat of bit 23, the sign bit of the two's complement number.

The user must ensure that integer operands conform to this format: integer results are automatically sign-extended to match.

FIX

The fix function converts a number from floating point format to sign-extended 24-bit two's complement integer format.

If the magnitude of its operand is greater than 2^{22}, $Encn_{1..0}$ is set to (0, 1), and $M_1 = 1$, it will set the Condition Register to 1. This limit was chosen to allow software to test for the case $n = 2^{23}$, which cannot be represented by a 24-bit two's complement number. If the operand is too large to be represented in the integer format, the result is clamped to either #007FFFFF or #FF800000, according to its sign.

fix does not attempt to set the Zero or Status Registers. It executes in the same number of cycles as every other multiplier/accumulator instruction.

FLOAT

The float function converts a number from sign-extended two's complement integer format to floating point format.

If its operand does not have consistent sign extension (bits 24–31 all equal), $Encn_{1..0}$ is set to (0, 1) and $M_1 = 1$, it will set the Condition Register to 1. The result of a float operation on such an operand is not defined.

float does not attempt to set the Zero or Status Registers. It executes in the same number of cycles as any other multiplier/accumulator instruction.

IEEE Considerations

The WTL 3132 and WTL 3332 comply with the IEEE Standard for Binary Floating Point Arithmetic (P754) in most respects. The differences described below apply to all of the arithmetic functions (fsubr, fsub, fadd, fmna, fmns, fmac, fabs).

DENORMALIZED NUMBERS

Denormalized numbers have a magnitude of less than 2^{-126} but greater than zero. The IEEE standard includes denormalized numbers to allow gradual underflow for operations that produce results that are too small to be expressed as normalized numbers. The WTL 3132/WTL 3332 do not support denormalized numbers. If the result of an operation is smaller than 2^{-126}, it is replaced by zero and the Zero Register is set to 1. Denormalized operands are detected and flushed to zero (with the same sign) before the operation is performed; no indication of this is provided.

NOT A NUMBER (NAN) HANDLING

The IEEE standard represents NaNs with numbers that have the maximum exponent value and a non-zero fraction. The WTL 3132/WTL 3332 do not detect attempts to perform calculations on NaNs. Only the flut operation may produce a NaN (when given zero as an operand). This is clamped to the appropriate infinity when refined by the divide code example given on page 38. Other operations may have undefined effects when given a NaN as an operand, so their use should, in general, be avoided. No arithmetic operation generates NaNs; all results with the maximum exponent have their fractions held to zero.

INFINITY AND OVERFLOW

The IEEE standard represents infinities with numbers that have the maximum exponent value and zero as the fraction. The WTL 3132/WTL 3332 do not detect attempts to perform calculations on infinite operands. Some operations may have undefined effects when given an infinite operand, so their propagation should, in general, be avoided. However, if an operation creates a result that is too large to be represented in the floating point format, its result is clamped to an infinite value as required by the specification. The Status Register is set by the creation of an infinite value during an operation.

UNDERFLOW

When the result of an operation has a magnitude in the range $0 < n < 2^{-126}$, the WTL 3132/WTL 3332 round it to zero and set the Zero Register to 1. There is no way to distinguish underflow from a result that is exactly zero.

ROUNDING

The WTL 3132/WTL 3332 support only the round-to-nearest mode: the infinitely precise result of an operation is rounded to the closest representation that fits in the destination format. If the result is exactly halfway between two representations, it is rounded to the nearest even fraction.

The IEEE standard requires rounding to occur after each arithmetic operation. The WTL 3132/WTL 3332 do not round between the multiply and add components of the fmac, fmns and fmna functions. The error in the result is always less than three least-significant bits.

If the ABin port of the ALU is set to the constant 0.0, then the fmac function performs a multiply that conforms to the IEEE standard. The fix operation only can be set to round to negative infinity by clearing M_1 to zero.

DC Specifications

ABSOLUTE MAXIMUM RATINGS

Supply voltage	-0.5 to 7.0 V
Input voltage	-0.5V to V_{DD}
Output voltage	-0.5V to V_{DD}
Operating temperature range (T_{CASE})	-55° C to 125° C
Storage temperature range	-65° C to 150° C
Lead temperature (10 seconds)	300° C
Junction temperature	175° C

RECOMMENDED OPERATING CONDITIONS

PARAMETER		COMMERCIAL MIN	COMMERCIAL MAX	UNIT
V_{DD}	Supply voltage	4.75	5.25	V
T_{CASE}	Operating temperature	0	85	°C

DC ELECTRICAL CHARACTERISTICS

PARAMETER		TEST CONDITIONS	COMMERCIAL (Note 1) MIN	TYP	MAX	UNIT
V_{IHC}	High level clock input voltage	V_{DD} = MAX	2.4			V
V_{ILC}	Low level clock input voltage	V_{DD} = MIN			0.8	V
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_{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_{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_{OZL}	Tri-state leakage current low	V_{DD} = MAX, V_{IN} = 0V			10	µA
I_{OZH}	Tri-state leakage current high	V_{DD} = MAX, V_{IN} = V_{DD}			10	µA
I_{DD}	Supply current	V_{DD} = MAX, T_{CY} = MIN TTL inputs (Note 2)			200	mA
C_{IN}	Input capacitance (Note 3)	V_{DD} = 5.0V		10	10	pF
C_{CLK}	Clock capacitance (Note 3)	$T_{AMBIENT}$ = 25°C		25	30	pF
C_{OUT}	I/O, Output capacitance (Note 3)	f = 1 MHz		15	20	pF
C_{OE}	OEX-, OEZ- capacitance (Note 3)			20	25	pF

Notes: 1. Worst case over power and temperature range
2. Input levels are 0.4V and 3.4V
3. Not tested

Timing Diagrams

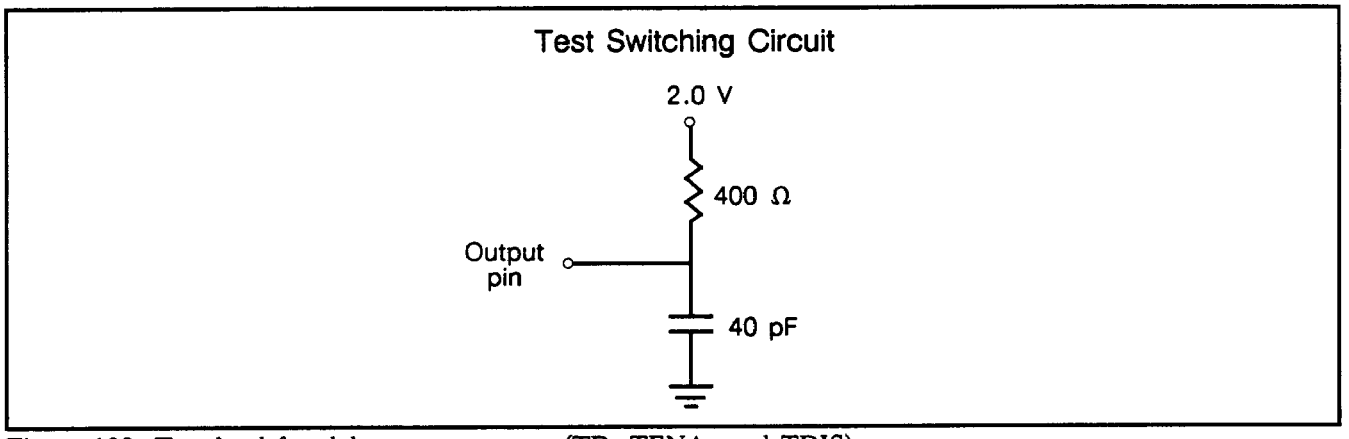


Figure 132. Test load for delay measurement (TD, TENA, and TDIS)

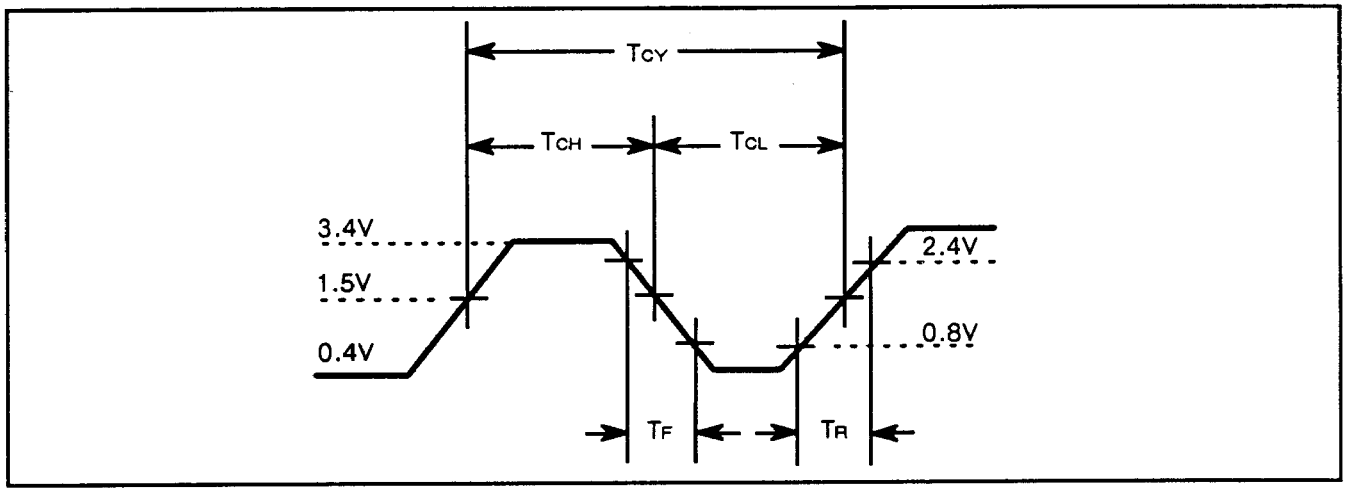


Figure 133. Clock timing

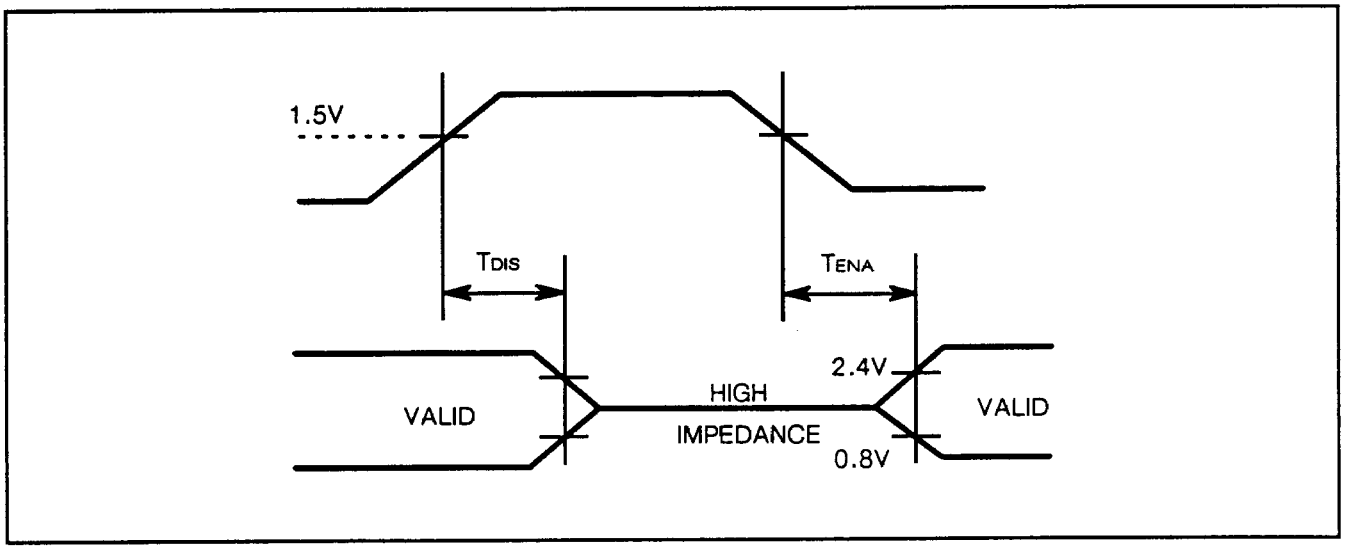


Figure 134. Tri-state timing

Timing Diagrams, continued

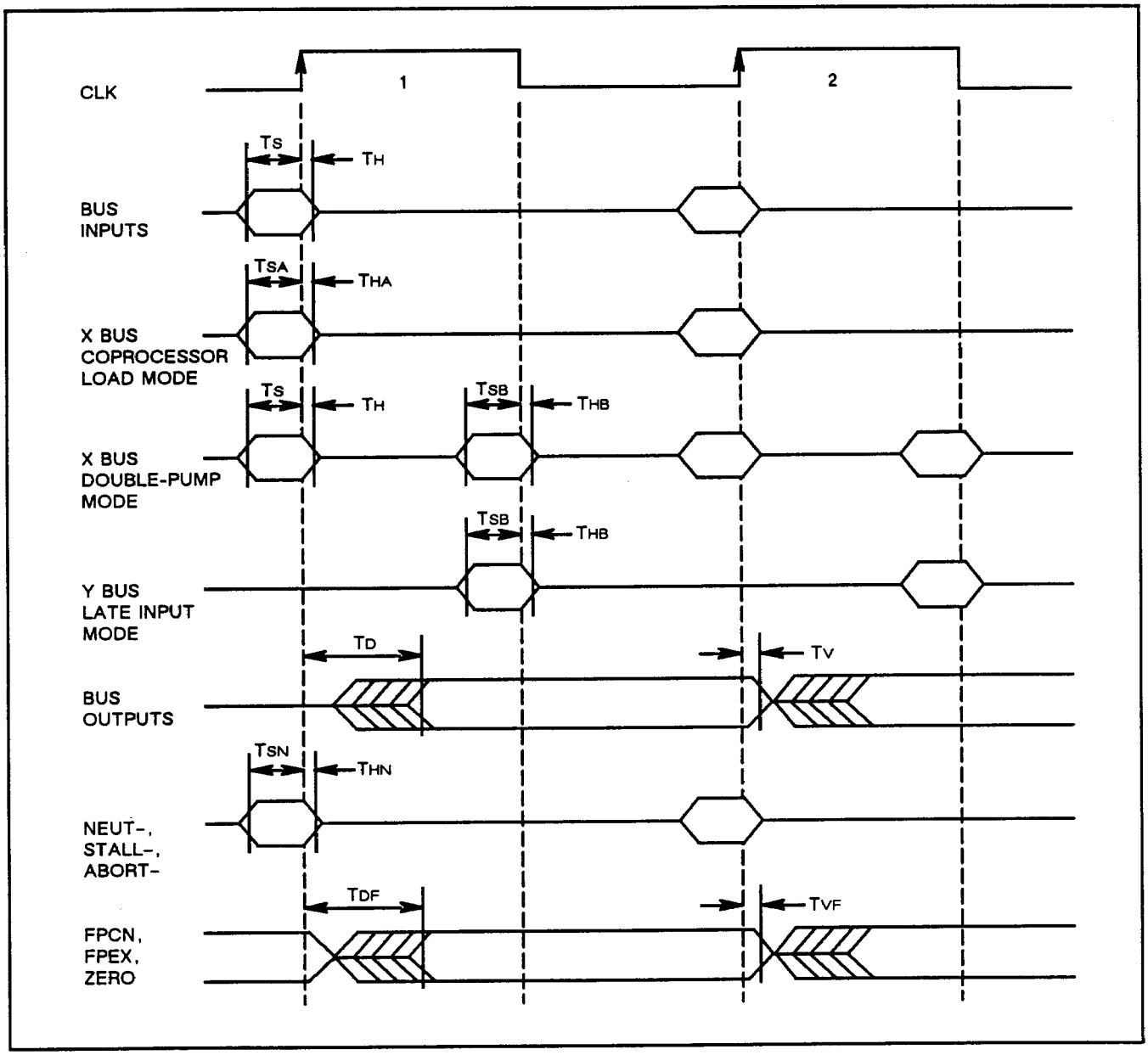


Figure 135. Signal timing diagram

October 1988

AC Specifications

AC SWITCHING CHARACTERISTICS

AC TEST CONDITIONS (Notes 1, 2, 3)				
V_{CC} = MIN	V_{IH} = 3.4V V_{IL} = 0.4V	V_{OH} = 2.8V, I_{OH} = -1.0 mA V_{OL} = 0.4V, I_{OL} = 4.0 mA	T_{CASE} = 85 °C	C_{LOAD} = 40 pF

DESCRIPTION	WTL 3132-120 WTL 3332-120 XL-3132-120 COMMERCIAL		WTL 3132-100 WTL 3332-100 XL-3132-100 COMMERCIAL		UNIT
	MIN	MAX	MIN	MAX	
T_{CY} Clock cycle time	120		100		ns
T_{CH} Clock high time	50		45		ns
T_{CL} Clock low time	50		45		ns
T_R Clock rise time (Note 5)		5		5	ns
T_F Clock fall time (Note 5)		5		5	ns
Bus inputs (C, X, Y):					
T_S Input setup time	20		15		ns
T_H Input hold time	2		2		ns
T_{SA} Input setup time	20		15		ns
T_{HA} Input hold time	2		2		ns
(X bus, coprocessor load mode)					
T_{SB} Input setup time	20		15		ns
T_{HB} Input hold time	2		2		ns
(X bus, double-pump or Y bus, late input mode)					
Bus outputs (X, Z):					
T_D Output delay time		35		30	ns
T_V Output valid time	3		3		ns
T_{ENA} Tri-state enable time (Note 4)		35		30	ns
T_{DIS} Tri-state disable time (Note 5)		35		30	ns
NEUT-, STALL-, ABORT-:					
T_{SN} Input setup time	20		15		ns
T_{HN} Input hold time	2		2		ns
FPCN, FPEX, ZERO:					
T_{DF} Output delay time		35		30	ns
T_{VF} Output valid time	3		3		ns
T_{OP} Pipelined operation time per stage	120		100		ns
T_{LA} Total latency register-to-register	360		300		ns

Notes:
1. Worst case over time and temperature range.
2. Input levels are 0.4 and 3.4V.
3. Timing transitions are measured at 1.5V unless otherwise noted.
4. Device must be powered for at least 20 ms before testing.
5. Not tested but guaranteed by design.

Figure 136. AC test conditions

Pin Configuration

Pin #1 Identifier	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	GND	NC	NC	X18	NC	NC	NC	X23	VDD	NC	NC	X27	NC	NC	NC
B	NC	NC	GND	X16	NC	NC	X20	X22	X24	X25	X26	NC	X29	X31	GND
C	NC	NC	X15	GND	X17	X19	X21	NC	VDD	NC	X28	NC	X30	TIE LOW	F2
D	NC	X12	X14										GND	F0	Adst1
E	X10	X11	X13										F1	Adst0	Abin0
F	X8	NC	NC										Abin2	Abin1	ABORT-
G	NC	X9	VDD										STALL-	NEUT-	Cwen-
H	NC	X7	VDD				WTL 3132 TOP VIEW						Cadd2	Cadd4	Cadd3
J	X6	X5	NC										Cadd1	Aadd3	Cadd0
K	NC	NC	NC										Aadd1	Aadd2	Aadd4
L	X4	X2	NC										Badd0	Badd4	Aadd0
M	X3	X1	GND										Dadd2	Badd1	Badd3
N	NC	NC	OEX-	VDD	FPCN	GND	TIE LOW	TIE LOW	GND	GND	GND	VDD	Dadd1	Dadd4	Badd2
P	X0	TIE LOW	Encn1	FPEX	Encn0	TIE LOW	CLK	GND	GND	GND	GND	GND	VDD	Dadd0	Dadd3
R	ZERO	GND	IOCt0	IOCt1	Mbin-	TIE LOW	TIE LOW	GND	GND	GND	GND	GND	GND	GND	GND

Notes: Pins marked "Tie Low" must be connected to ground. Pins marked "NC" should be left unconnected (floating).

Figure 137. WTL 3132 and XL-3132 pin configuration

Pin Configuration, continued

Pin #1 Identifier	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A	OEX-	TIE LOW	Z0	Z2	X3	Z4	NC	X5	Z7	X9	Z9	X11	Z12	X13	X15	Z15	GND
B	OEZ-	ZERO	X0	Z1	X2	NC	NC	Z5	VDD	VDD	X8	Z10	Z11	Z13	X14	GND	GND
C	GND	Encn1	GND	X1	Z3	X4	X6	NC	Z6	Z8	X7	X10	X12	Z14	Z16	X17	Z17
D	IOCt0	FPEX	FPCN												X16	Z18	X18
E	Encn0	Mbin-	IOCt1												Z20	X19	Z21
F	Mbs-	CLK	TIE LOW												Z19	X21	X20
G	TIE LOW	TIE LOW	Y28												VDD	VDD	VDD
H	Y31	TIE LOW	Y30												X23	X22	Z22
J	TIE LOW	Y29	Y26												Z23	X24	Z24
K	Y22	Y24	Y25												Z25	X25	Z26
L	Y23	Y21	Y27												VDD	VDD	VDD
M	Y18	Y19	Y20												X27	X28	Z27
N	Y16	Y15	Y17												X29	Z28	X26
P	Y14	Y13	Y12												Z31	Z30	Z29
R	Y10	Y11	Y9	Dadd3	Y6	Badd2	Aadd4	Aadd1	Cadd0	Cadd1	Y2	Y0	STALL-	Adst1	F1	X31	X30
T	VDD	VDD	Dadd2	Y7	Badd1	Badd3	Aadd0	Aadd2	Aadd3	Cadd2	Cadd4	Cwen-	Abin0	Abin2	Adst0	F0	TIE LOW
U	Y8	Dadd0	Dadd1	Dadd4	Badd0	Y5	Badd4	Y4	Y3	Y1	Cadd3	ABORT-	NEUT-	Abin1	F2	GND	GND

WTL 3332
TOP VIEW

Notes: Pins marked "Tie Low" must be connected to ground. Pins marked "NC" should be left unconnected (floating).

Figure 138. WTL 3332 pin configuration

Physical Dimensions

WTL 3132 144-PIN PIN GRID ARRAY

Symbol	DIMENSIONS	
	INCHES	MM
A1	0.080 ± 0.008	2.03 ± 0.20
A2	0.180 typ.	4.57 typ.
A3	0.050	1.27
D	1.575 sq. ± 0.016	40.0 sq. ± 0.41
E1	1.400 sq. ± 0.012	35.56 sq. ± 0.30
E2	0.050 dia. typ.	1.27 dia. typ.
E3	0.018 ±0.002	.46 ± 0.05
d	0.070 dia. typ.	1.78 dia. typ.
e	0.100 typ.	2.54 typ.

WTL 3332 168-PIN PIN GRID ARRAY

Symbol	DIMENSIONS	
	INCHES	MM
A1	0.095 ± 0.009	2.41 ± 0.23
A2	0.180 typ.	4.57 typ.
A3	0.050	1.27
D	1.750 sq. ± 0.018	44.5 sq. ± 0.46
E1	1.600 sq.	40.6 sq.
E2	0.050 dia. typ.	1.27 dia. typ.
E3	0.018 ±0.002	.46 ± 0.05
d	0.070 dia. typ.	1.78 dia. typ.
e	0.100 typ.	2.54 typ.

Appendix A: The XL-3132 in the XL Environment

WEITEK XL-SERIES

The WEITEK XL-Series is a family of three VLSI processors: the XL-8000, a high-speed 32-bit integer processor; the XL-8032, a single-precision floating point processor; and the XL-8064, a double-precision floating point processor.

These processors give the performance of bit-slice components, and are supported by a full complement of development tools. These include C and FORTRAN 77 compilers, and an assembler which produces code that may be optimized by an instruction parallelizer. The development system offers both hardware and software simulators with debugging facilities. The programmer remains free to create custom microcode routines for peak performance.

This appendix is dedicated to the XL-8032 single-precision floating point processor. Further information may be found in the *XL-Series Overview*, the *XL-Series Hardware Designer's Guide*, the *XL-Series Programmer's Reference Manual*, the *XL-8136 Data Sheet* and the *XL-8137 Data Sheet*.

The XL-8032 processor consists of three interconnected VLSI components:

- XL-8136 Program Sequencing Unit (PSU)
- XL-8137 Integer Processing Unit (IPU)
- XL-3132 Floating Point Unit (FPU)

Each of these components is manufactured in high-density, low-power CMOS and delivered in a 144-pin grid array package. Unlike a traditional microprocessor, the XL-8032 is not constrained by the limits of circuit density or bus bandwidth imposed by a single chip in a small package. Consequently, bit-slice performance levels can be obtained both for integer and floating point operations.

The XL-Series simplifies system design. Zero glue interfacing is provided by a small number of dedicated signals that communicate state information between the components. These signals and the system buses need only be connected as shown in figure 69 in order to create the XL-8032. The purpose of each interconnection is described in more detail below.

BUSES

Four high-bandwidth system buses are provided by the XL-8032:

1. Code bus.

   The 64-bit code bus feeds the code input ports of the PSU, IPU and FPU. The PSU and IPU share 32 of the 64-bits; this half of the code word directs program control operations, address generation, loads and stores, and integer arithmetic. The remainder of the code word directs floating point operation. The designer may choose to lengthen the code word to add custom extensions to the processor architecture.

2. Data bus.

   The 32-bit data bus is shared by the IPU and FPU. It allows bytes, 32-bit integers and 32-bit floating point numbers to be transferred between the processing units and data memory.

3. Code Address bus.

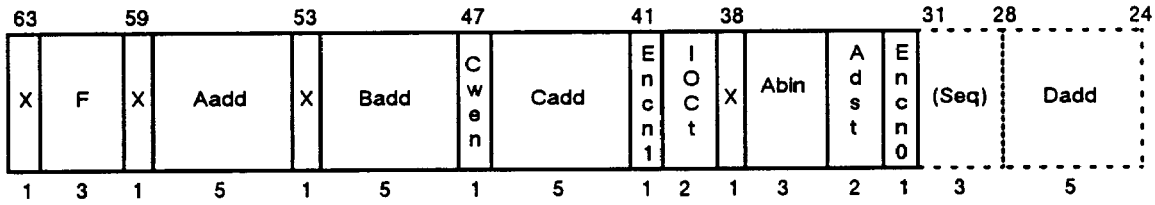
   The 32-bit code address bus carries the address of the next instruction from the PSU to the code memory. A word address allows up to 32 Gbytes of 64-bit wide code memory.

4. Data Address bus.

   A 32-bit data address bus carries the address of the next data read or write. The address is generated by the IPU and the data may be transferred to or from the IPU or FPU as required. A byte address allows up to 4 Gbytes of 32-bit wide data memory. Support for accessing bytes, half-words, and words is provided by the IPU.

The XL-3132 is designed to hook directly to the code and data buses alongside the other components of the XL-8032. When driven by the same system clock, the code word is sampled by all three components simultaneously, and the data bus is driven or sampled at the same time in the cycle no matter which component is transferring information.

The code and data memory systems may be implemented with SRAM, static column DRAM, or interleaved DRAM. Both code and data caches are supported by the XL-8032 chip set. More details are provided in the *XL-Series Hardware Designer's Guide*.



1. Dashed lines indicate bits in the sequencer field.
2. Bits marked with an "X" are reserved bits.
3. The meaning of each bit field is described in the body of this data sheet.

Figure 139. XL-3132 signal assignments in the XL-8032 code word

INSTRUCTION FORMAT

The XL-8032 has a 64-bit instruction word. The bits that are directed to the XL-3132 are shown in figure 66.

The lower 32-bits of the instruction word are shared by the IPU and the PSU. Bits 0–23 normally define the IPU operation. Bits 24–31 define the instruction flow control performed by the PSU. Five of these control bits, 24–28, are also used as the floating point register address (Dadd) when floating point load and store operations are performed. This saves on code bits and insures that the FPU and IPU never compete for the data bus.

The XL-3132 has 34 C port inputs. When used in the XL-8032 configuration, the Dadd field is tied to the appropriate bits of the PSU code input. The Mbin– bit is tied to GND, reducing the number of bits used in the XL-8032 instruction word to 60; the remaining four are marked with an X in figure 66 and should be set to 0 in any code to maintain compatibility with future versions of the processor.

LOAD/STORE MODEL

The XL-Series has a consistent load/store model regardless of processor configuration. Each processing unit has its own register file: register moves between the IPU and FPU must be made through the data memory. Each of these register files is multi-ported and each register may be the operand source or result destination of any instruction implemented by the unit. When an instruction takes more than one cycle to execute, the registers that supply its operands and receive its result cannot be modified until it has been completed. This allows any instruction to be resubmitted for execution after an interrupt with its original state.

Transactions between the register files and data memory are performed with dedicated load/store instructions. The only restriction on loads and stores is that the operands of an operation be loaded before it is executed and that it shall have completed before its result is stored. This allows the parallelizer considerable freedom to optimize register usage and I/O transactions.

An example of the normal sequence of operation is given below. This example leaves several free cycles in which other loads, stores, and calculations could be performed in parallel.

```
.
.
addr .ra
fload .fx
fabs .fx, .fy
nop
addr .rb
fstore .fy
.
.
```

Figure 140. * rb = |*ra|

Appendix A: The XL-3132 in the XL Environment, continued

Using the Coprocessor Load Mode gives the fload and fstore operations the same timing as the IPU's load and store operations. For loads, the address is presented on the AD bus at the beginning of a cycle and the data is expected to be available on the D bus by the end of that cycle. For stores, the address is presented on the AD bus at the beginning of a cycle and the data is driven onto the D bus during the *next* cycle.

Because the addr and fload instructions can be executed in parallel, they may be pipelined to support contiguous load operations (one per cycle).

If, however, loads and stores are to be interleaved, each store must be allowed two cycles; the latter cycle is then always available for the I/O nop required when following an fstore by an fload. This has minimal impact on overall performance because loads usually outnumber stores; and the parallelizer can organize I/O transfers efficiently. If the code constraints covered in the body of this data sheet are followed or if WEITEK software tools are used, then this load/store model will be obeyed.

MODES

The XL-3132 Mode Register must be initialized to the values given in figure 68 when coupled into the XL-8032 processor. The resulting programming model is illustrated in figure 70. Each selection is explained here:

1. Internal Bypass Mode is enabled to maximize performance.
2. The fix and float range test may be enabled or disabled as required.
3. Input Bypass (and Double-Pump) Modes are disabled, because they cannot operate when the Coprocessor Load Mode is enabled. The MBin and ABin ports on the multiplier/accumulator cannot, therefore, receive operands from the C bus.
4. Output Bypass Mode is enabled to maximize performance.
5. Overflows may be enabled or disabled as required.
6. Coprocessor Load Mode is enabled so that the XL-3132 matches the XL-Series load/store model.
7. The Y port Late Input Mode is disabled; it only operates with the WTL 3332.

MODE BIT	LOGIC VALUE	DESCRIPTION
M_0	1	Internal Bypass Mode (Aadd = Cadd) enabled
M_1	1	fix and float range test enabled (may be disabled)
M_2	0	Reserved: must be cleared to 0
M_3	0	Input Bypass Mode disabled
M_4	1	Output Bypass Mode enabled
M_5	1	Overflow exception enabled (may be disabled)
M_6	1	Coprocessor Load Mode enabled
M_7	1	Reserved: must be set to 1
M_8	0	FPEX active low and "sticky"*
M_9	0	Double-pump Mode disabled
M_{10}	1	Reserved: must be set to 1
M_{11}	1	Internal Bypass Mode (Aadd = Badd) enabled
M_{12}	0	Y Late Input Mode disabled

*These features are not available on all versions of the XL-3132; check the *Programmer's Reference Manual* for details

Figure 141. Mode selection table

Appendix A: The XL-3132 in the XL Environment, continued

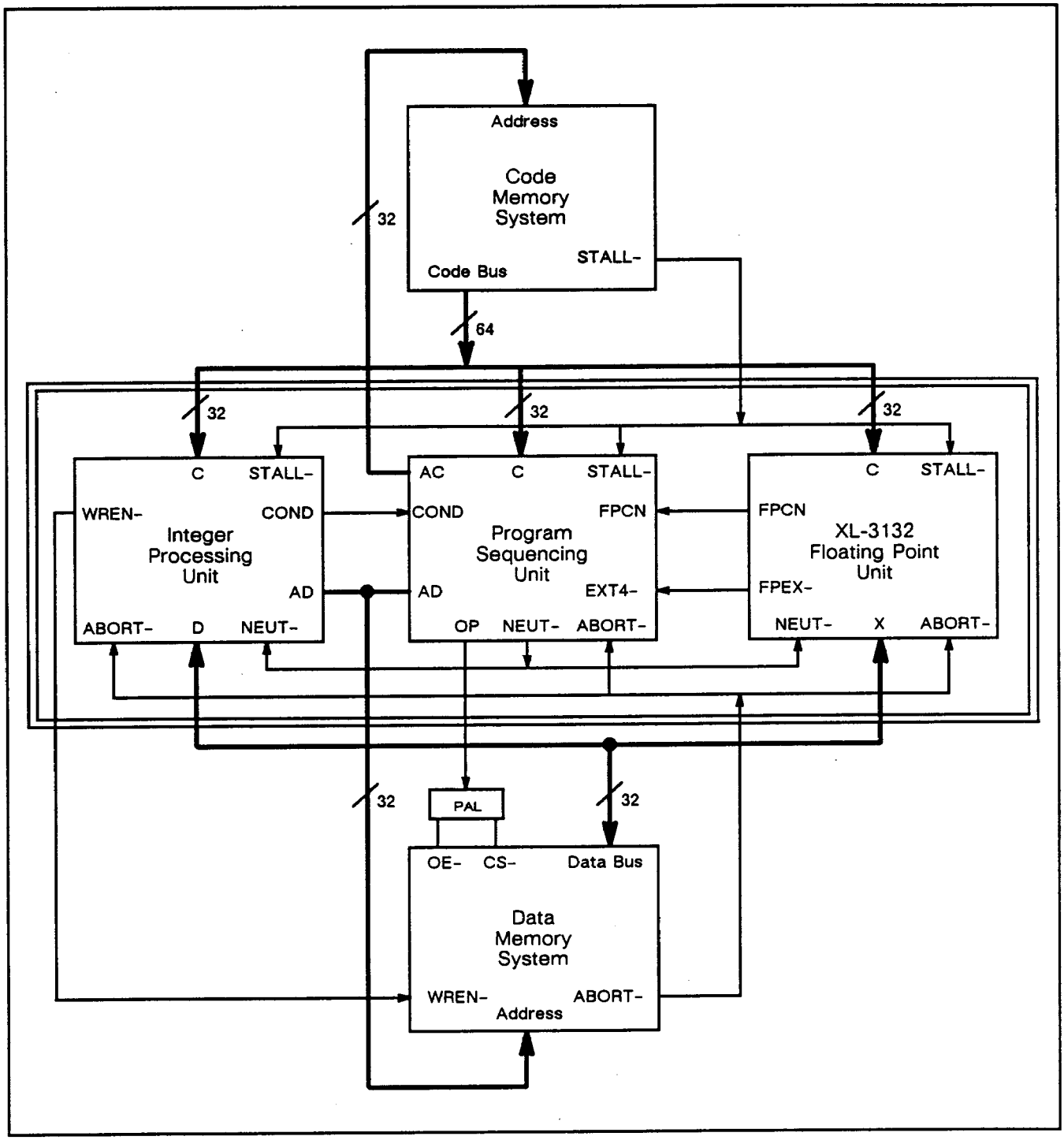


Figure 142. XL-8032 schematic

Appendix A: The XL-3132 in the XL Environment, continued

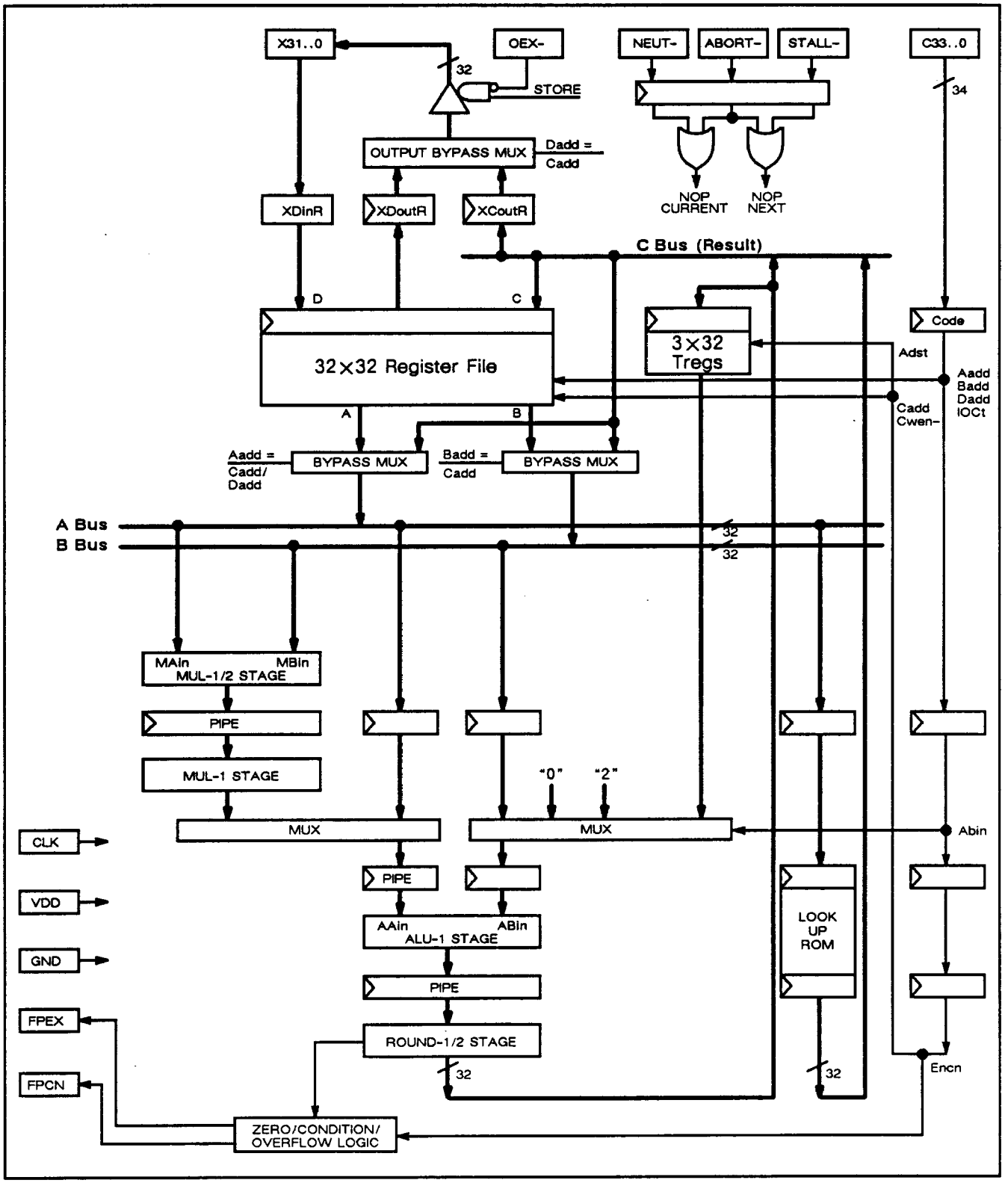


Figure 143. XL-3132 in XL mode

CONDITIONS AND EXCEPTIONS

The XL-8032 provides several signals which transfer state information from the processing units (IPU, FPU) to the sequencer (PSU). These are either conditions, upon which the PSU may decide to branch; or exceptions, which require software intervention to recover gracefully.

The FPCN output on the XL-3132 should be connected to the FPCN input on the XL-8136. The FPCN signal is enabled by the $Encn_{1..0}$ field in the instruction word to indicate whether the result of an operation is = 0, < 0, or <= 0. The PSU may then execute a "branch on condition" instruction to selectively transfer program control according to the outcome of this comparison.

The FPEX- output on the XL-3132 should be connected the EXT4- interrupt input on the XL-8136. If the overflow enable bit in the mode register is set, then any arithmetic operation that generates an invalid result can flag this exception to the PSU. The system software is expected to react appropriately to this interrupt.

NEUT-, STALL- AND ABORT-

The XL-8032 components all use the NEUT-, STALL- and ABORT- signals. These pins should be connected directly between the three chips in the XL-8032 processor (see figure 69).

NEUT- cancels the effect of the current instruction. The signal is generated by the PSU. It is normally used in the shadow of a delayed branch to prevent the instruction in the pipeline from having any effect on the state of the IPU and FPU.

STALL- cancels the effect of the next instruction. It should be generated by the code memory subsystem to indicate the delay or absence of the correct code word. This prevents any invalid operation that may be present on the code input at this time from affecting the state of the processor. It allows wait states to be inserted in code fetches, perhaps to allow for DRAM refresh or a code cache miss.

ABORT- cancels the effect of both the current and the next instructions. It should be generated by the data memory subsystem to indicate the inability of the system to instantly access the required data word. This allows the canceled instructions to be repeated when the address becomes valid and for this retry to have the correct effect. It allows the data memory to be "not ready" if, for example, a page fault occurs.

Appendix B: Programming Examples

In the following examples, square root has Coprocessor Load Mode enabled and Input Bypass Mode disabled. It may be used in the XL-Series environment. The other examples have Coprocessor Load Mode disabled and Input Bypass Mode enabled. They must be modified to be run on the XL-Series machines.

SQUARE ROOT

NOTATION:

a = operand (.f4)

R_0 = seed (.f0)

R_1 = first approximation (.f1)

R_2 = second approximation (.f2)

0.5 = constant (.f5)

3.0 = constant (.f6, .t1)

s = result (.f3)

ALGORITHM:

$$R_n = \frac{R_{n-1}}{2}\ 3 - a R_{n-1}^2$$

$$\sqrt{a} \simeq aR_n$$

$$R_1 = \frac{R_0}{2}\ 3 - a R_0^2$$

$$R_2 = \frac{R_1}{2}\ 3 - a R_1^2$$

Cycle #	Opcode					I/O	Comment
1	fnop				;	fload .f6	
2	fadd	.f6	0,	.t1	;	fload .f0	move 3.0 to .t1
3	fmac	.f0,	.f0,	0,	.f6;	fload .f5	$R_0 \times R_0$
4	fmac	.f0,	.f5,	0,	.f3;	fload .f4	$0.5 \times R_0$
5	fnop				;		
6	fmna	.f4,	.f6,	.t1,	.f7		$3.0 - (a \times R_0^2)$
7	fnop						
8	fnop						
9	fmac	.f3,	.f7,	0,	.f1		R_1
10	fnop						
11	fnop						
12	fmac	.f1,	.f1,	0,	.f6		$R_1 \times R_1$
13	fmac	.f1,	.f5,	0,	.f3		$0.5 \times R_1$
14	fnop						
15	fmna	.f4,	.f6,	.t1,	.f7		$3.0 - (a \times R_1^2)$
16	fnop						
17	fnop						
18	fmac	.f3,	.f7,	0,	.f2		R_2
19	fnop						
20	fnop						
21	fmac	.f2,	.f4,	0,	.f3		$s = a \times R_2$
22	fnop						
23	fnop						
24	fnop				;	fstore .f3	store s
25	fnop						s on X port

Figure 144.

Appendix B: Programming Examples, continued

The square root code given fetches a seed for the first approximation to the result and then proceeds to refine its value using the *Newton-Raphson* algorithm. It differs from the example for division given in the body of the data sheet in that no on-chip look-up table is provided for the seed value.

The external table assumed here has the same accuracy as the internal divide look-up table (all of the exponent and the seven most-significant bits of the fraction). The following formulae may be used to calculate the entries of this table.

Table entry exponent (G) and fraction (H) in terms of the operand values (E) and (F). Care should be taken to insure that zeroes, negative numbers, infinities and NaNs are handled correctly.

$$G = \left[\frac{380 - E}{2}\right]$$

$$\text{if } E(8) = 1; \quad H = \left[\frac{2^{13}}{\sqrt{257 + F}}\right] - 256$$

$$\text{if } E(8) = 0; \quad H = \left[\frac{2^{13}}{\sqrt{512 + 2F}}\right] - 256$$

[ ] = truncate to lower integer.

4×4 MATRIX TRANSFORM

NOTATION:

Transform matrix	A (a_{11} to a_{44}) (.f16 to .f31)
Operand vector	X (x_1 to x_4) (.f0 to .f3)
Result vector	Y (y_1 to y_4) (.f4 to .f7)
Partial results	p_1 to p_4 (Tregs)

ALGORITHM:

$Y = AX$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

Cycle #	Opcode					I/O	Comment
1	fmac	.f0,	.f16,	0,	.t1;	fload .f0	$p_1 = a_{11} \times x_1$
2	fmac	.f0,	.f17,	0,	.t2		$p_2 = a_{21} \times x_1$
3	fmac	.f0,	.f18,	0,	.t3		$p_3 = a_{31} \times x_1$
4	fmac	.f0,	.f19,	0,	.t1		$p_4 = a_{41} \times x_1$
5	fmac	.f1,	.f20,	.t1,	.t2;	fload .f1	$p_1 = a_{12} \times x_2 + p_1$
6	fmac	.f1,	.f21,	.t2,	.t3		$p_2 = a_{22} \times x_2 + p_2$
7	fmac	.f1,	.f22,	.t3,	.t1		$p_3 = a_{32} \times x_2 + p_3$
8	fmac	.f1,	.f23,	.t1,	.t2		$p_4 = a_{42} \times x_2 + p_4$
9	fmac	.f2,	.f24,	.t2,	.t3;	fload .f2	$p_1 = a_{13} \times x_3 + p_1$
10	fmac	.f2,	.f25,	.t3,	.t1		$p_2 = a_{23} \times x_3 + p_2$
11	fmac	.f2,	.f26,	.t1,	.t2		$p_3 = a_{33} \times x_3 + p_3$
12	fmac	.f2,	.f27,	.t2,	.t3		$p_4 = a_{43} \times x_3 + p_4$
13	fmac	.f3,	.f28,	.t3,	.f4;	fload .f3	$y_1 = a_{14} \times x_3 + p_1$
14	fmac	.f3,	.f29,	.t1,	.f5		$y_2 = a_{24} \times x_3 + p_2$
15	fmac	.f3,	.f30,	.t2,	.f6		$y_3 = a_{34} \times x_3 + p_3$
16	fmac	.f3,	.f31,	.t3,	.f7		$y_4 = a_{44} \times x_3 + p_4$

Note: This example is not interruptable. Results can be read from the Z port of the WTL 3332

Figure 145.

Appendix B: Programming Examples, continued

RADIX-2 FFT

In-place for algorithm for radix-2 butterflies.
Two butterflies are evaluated per iteration.

NOTATION:

Source Operands	*Results*
A_r, $A_r\#$ (.f4, .f20)	A_r', $A_r\#'$
A_i, $A_i\#$ (.f5, .f21)	A_i', $A_i\#'$
B_r, $B_r\#$ (.f3, .f19)	B_r', $B_r\#'$
B_i, $B_i\#$ (.f0, .f16)	B_i', $B\#'$
W_r, $W_r\#$ (.f2, .f18)	
W_i, $W_i\#$ (.f1, .f17)	

ALGORITHM:

$A_r' = A_r + (B_r \times W_r - B_i \times W_i)$

$A_i' = A_i + (B_r \times W_i + B_i \times W_r)$

$B_r' = A_r - (B_r \times W_r - B_i \times W_i)$

$B_i' = A_i - (B_r \times W_i + B_i \times W_r)$

Cycle #	Opcode					I/O	Comment
	fnop					; fload .f0	
	fnop					; fload .f16	
1	fmns	.f1,	.f0,	0,	.t1	; fload .f1	$-B_i \times W_i$
2	fmac	.f2,	.f0,	0,	.t2	; fload .f2	$+B_i \times W_r$
3	fmns	.f17,	.f16,	0,	.t3	; fload .f17	$-B_i\# \times W_i\#$
4	fmac	.f18,	.f16,	0,	.t1	; fload .f18	$+B_i\# \times W_r\#$
5	fmac	.f3,	.f2,	.t1,	.f8	; fload .f3	$B_r \times W_r - B_i \times W_i$
6	fmac	.f3,	.f1,	.t2,	.f9		$B_r \times W_i + B_i \times W_r$
7	fmac	.f19,	.f18,	.t3,	.f24	; fload .f19	$B_r\# \times W_r\# - B_i\# \times W_i\#$
8	fmac	.f19,	.f17,	.t1,	.f25		$B_r\# \times W_i\# + B_i\# \times W_r\#$
9	fadd	.f4,	.f8,	.f10		; fload .f4	A_r'
10	fsub	.f4,	.f8,	.f11			B_r'
11	fadd	.f5,	.f9,	.f12		; fload .f5	A_i'
12	fsub	.f5,	.f9,	.f13			B_i'
13	fadd	.f20,	.f24,	.f26		; fload .f20	$A_r\#'$
14	fsub	.f20,	.f24,	.f27		; fload .f21	$B_r\#'$
15	fadd	.f21,	.f25,	.f28		; fload .f0	$A_i\#'$
16	fsub	.f21,	.f25,	.f29		; fload .f16	$B_i\#'$

Note: This example is not interruptable. Results can be read from the Z port of the WTL 3332

Figure 146.

Ordering Information

PACKAGE TYPE	TEMPERATURE RANGE	ORDER NUMBER
144-Pin PGA	Tc = 0 to +85 ° C	WTL 3132-GCD-100, -120
168-Pin PGA	Tc = 0 to +85 ° C	WTL 3332-GCD-100, -120
144-Pin PGA	Tc = -55 to +125 ° C	WTL 3132-GMD-100, -120
168-Pin PGA	Tc = -55 to +125 ° C	WTL 3332-GMD-100, -120

XL-Series customers should order the following:

PACKAGE TYPE	TEMPERATURE RANGE	ORDER NUMBER
144-Pin PGA	Tc = 0 to +85 ° C	XL-3132-GCD-100, -120
144-Pin PGA	Tc = -55 to +125 ° C	XL-3132-GMD-100, -120

October 1988

Revision Summary

CONTENTS

This table lists major changes since the October 1987 printing of this data sheet.

Change	Description
1. NEUT– or ABORT– and fmode	Revised, para 5 on page 12; page 29
2. Reverse C bus operation	Revised, new note, page 16
3. Register write WRI	Corrected, figure 21, page 19
4. D_3 A bus transfer	Corrected, figure 31, page 24
5. fabs condition on N = 0 when M_1 = 0	Revised, figure 32, page 25
6. Figure caption	Corrected, figure 47, page 33
7. Clarification of range bit	Revised, figure 52, page 35
8. Mode bit	Corrected, figure 54, page 37
9. Divide result precision	Revised, page 39
10. Encn settings for range text	Revised, page 40
11. Absolute maximum ratings	Revised, page 42

COMPONENTS

This table lists the component revision to which successive versions of this data sheet have referred.

Data	Date	Component Suffix
WTL 3132/WTL 3332 Preliminary Data	July 1986	A
XL-Series Product Status Report	March 1987	A
XL-Series Product Status Report	July 1987	C
WTL 3132/WTL 3332 Errata Sheet	October 1987	C
WTL 3132/WTL 3332 Data Sheet	October 1987	F
WTL 3132/WTL 3332 Data Sheet	October 1988	F