WEITEK

XL-8236 22-BIT RASTER CODE SEQUENCER

PRELIMINARY DATA
October 1988

The WEITEK XL-8236 is a fully-integrated CMOS 22-bit raster code sequencer. It is used with the WEITEK XL-8237 32-bit raster image processor to make the HyperScript-Processor™, a high-performance graphics CPU capable of driving raster printers at up to 60 pages per minute. WEITEK's single-precision floating point unit may also be used to produce a tightly-coupled raster image printing system.

WEITEK
2 Pm 10/1
TIME: DATE:
PRELIMINARY
TRANSMITTED BY:
Julie Cater

Contents

The masters for this document were printed on an XL-8200 development system

XL-8236 Raster Code Sequencer Data Sheet
October, 1988

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

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

XL-8236
22-BIT RASTER CODE SEQUENCER

PRELIMINARY DATA
October 1988

Features

22-BIT SINGLE-CHIP SEQUENCING UNIT

22-bit code address bus
32-bit data address bus
33×32-bit on-chip stack

HIGH PERFORMANCE

10 to 60 page per minute with WEITEK's HyperScript interpreter
Low-power CMOS with TTL-compatible I/O

POWERFUL DEVELOPMENT TOOLS

PostScript-compatible interpreter
C compiler
Graphics development system

BUILT-IN REGISTERS AND TIMERS

Breakpoint register
32-bit programmable timer
Status register

TRAP AND INTERRUPT HANDLING

Three external interrupt lines
Five internal exceptions
System reset

Description

The XL-8236 is a high-performance 22-bit raster code sequencer (RCS). The XL-8236 combines with its companion chip, the XL-8237 raster image processor (RIP), to make the XL-8200 HyperScript-Processor, a cost-effective graphics CPU for raster printing applications over a wide performance range. The most typical use of a HyperScript-Processor is in a PostScript-language laser printer.

HyperScript-Processors are graphics RISC processors that combine Harvard architecture, single-cycle instruction execution, and specialized math and bit manipulation functions to make a high-speed grapics processor, capable of interpreting the complex PostScript language on high-speed printers. The XL-8236 provides instruction sequencing and other control functions. The XL-8237 provides data addressing, arithmetic, logical, and bit-manipulation fuctions.

The XL-8236 is a CMOS device offering high performance and low power consumption, with TTL-compatible I/O. It is available in a standard 145-pin ceramic or plastic PGA (Pin Grid Array) package.

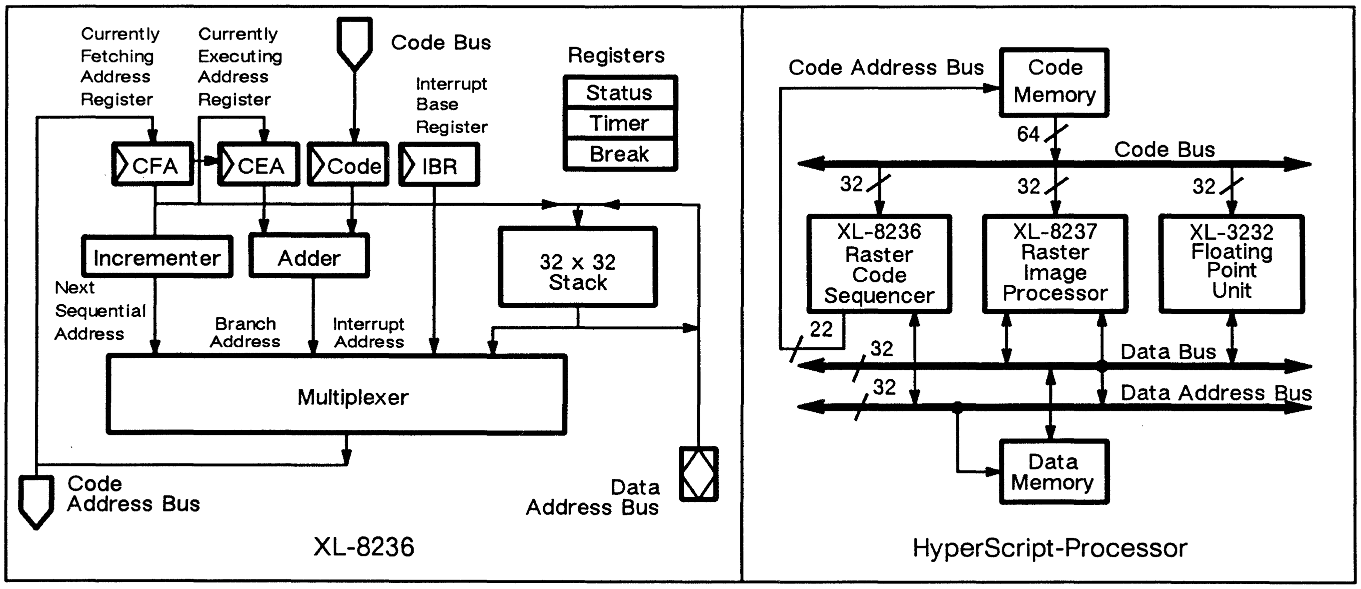


Figure 1. Simplified block diagrams

Block Diagram

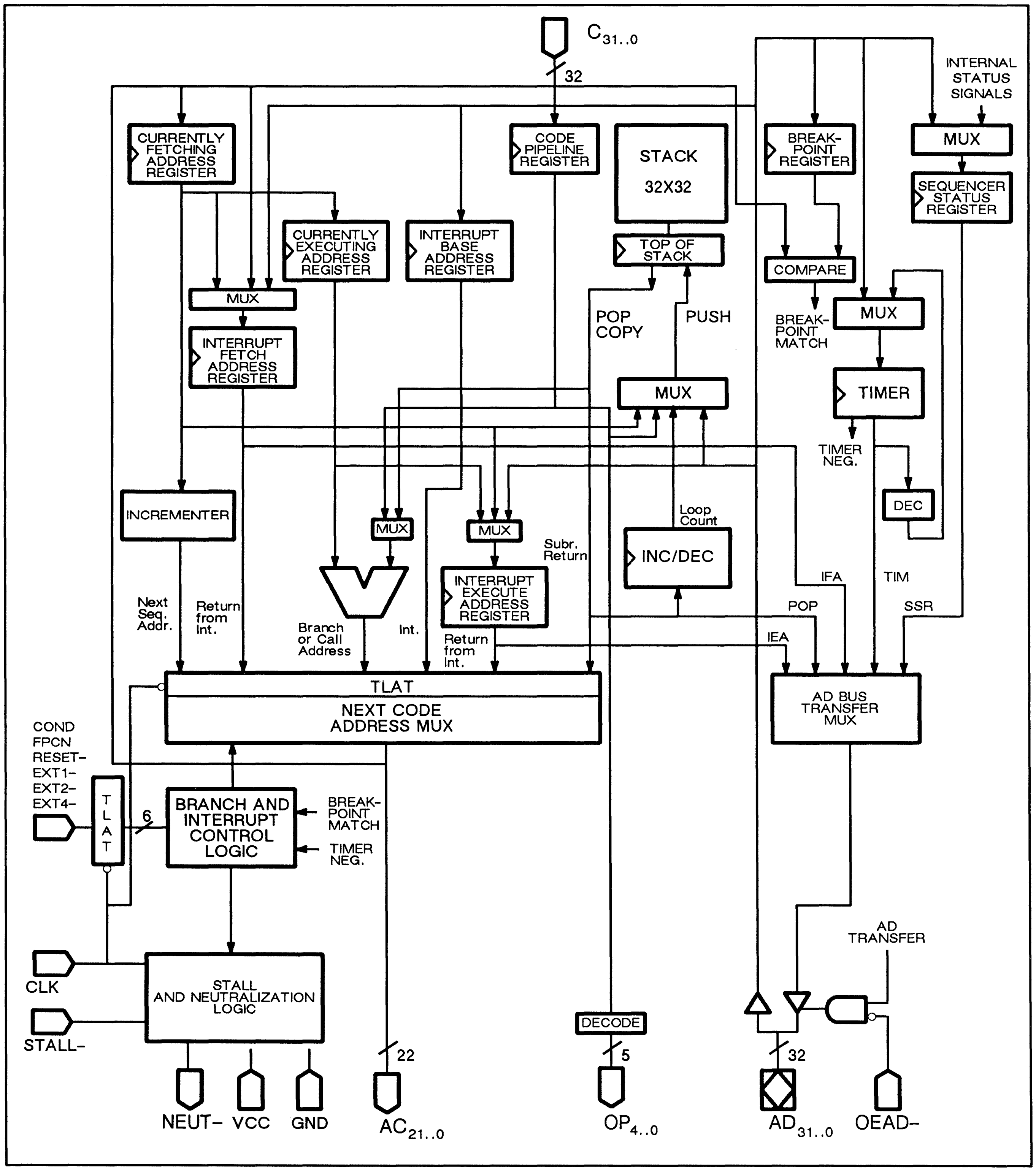


Figure 2. Block diagram

Signal Description

C BUS

The $C_{31..0}$ *code input bus* is driven by the code memory with the 32-bit instruction word. The code word is latched by the RCS at the rising edge of the clock. Because it contains a built-in pipeline register, it is not necessary to use an external pipeline register between code memory and the XL-8236.

AC BUS

The $AC_{21..0}$ *code address output bus* is driven by the program sequencing unit. It sends a 22-bit instruction address to the code memory. The code address is not latched by the RCS, so an external address latch is needed between the AC bus and code memory.* Unused high-order bits of the AC bus can be left floating.

The AC bus is driven on every cycle (even when STALL–, or NEUT– are asserted) unless disabled with the OEAC– signal.

The AC bus produces instruction addresses, not byte addresses.

AD BUS

The $AD_{31..0}$ *data address bus* provides addresses for data memory operations (the data is transferred over the D bus to the RIP). It is also used for intra-processor communication. It connects the integer processing unit to the sequencer. The AD bus can also be used as a bidirectional data bus for transfers to and from other hardware.

All 32 bits of the AD bus need to be attached between the RCS and RIP to allow intra-processor data transfer. This traffic may take place during STALL– or NEUT cycles, and it is important that it not be interfered with. If any external device wishes to write to memory asynchronously to the XL-Series devices, it must not write directly to the AD bus.

Addresses on the AD bus are byte addresses.

OP BUS

The $OP_{4..0}$ output bus indicates the type of instruction that is executing, and can be used to control external hardware. The memory system must decode the OP bus outputs to determine when to read, when to write, and when to latch the data address. In addition, fifteen of the 32 OP combinations are used to signal loads or stores to "external registers" 0–14, which can be any external hardware. These external register transfers take place over the AD bus.

* Note that a latch, not a register, should be used. Future versions of the XL-8236 may contain an on-chip address latch.

EXT1–, EXT2–, and EXT4–

Level-sensitive interrupt request inputs. The current instruction is allowed to complete and execution proceeds from one of the interrupt vectors. External interrupts can be enabled and disabled in the sequencer status register. Interrupt signals are examined at the rising edge of the clock.

EXT4– is used as a floating point exception interrupt in systems with the XL-3232 FPU.

There is no EXT3–.

Interrupt signals must be held until acknowledged.

RESET–

A level-sensitive input that resets the sequencer and causes a branch to address 0. The sequencer status register is initialized as described on page 18. The other registers in the chip are undefined. Registers that can cause exceptions (such as the timer and breakpoint registers) must be initialized before their exceptions are enabled.

Reset is not useful as a non-maskable interrupt.

CLK

The Clock input, CLK, is a single-phase TTL-level clock signal.

NEUT–

NEUT– (neutralize) is an output signal that goes from the RCS to the RIP and FPU. It is not normally used by hardware outside the processor chip set. NEUT– is asserted by the sequencer, and instructs all XL devices to cancel their current instructions. This is done on transfer-of-control instructions (including branches, calls, and interrupts) to prevent the instruction in the pipeline from being executed. All XL-Series chips must have their NEUT– lines tied together.

Signal Description, continued

STALL–

STALL– is a "not-ready" input line that causes the current code fetch to be retried on the next cycle. The instruction that was to be executed on the next cycle is canceled (but the current instruction is allowed to complete). STALL– is typically used by the code memory subsystem when the requested code word cannot be read in the current cycle.

The XL chips each cancel their currently fetching instruction, and fetch the instruction again on the next cycle, and on every cycle that STALL– is asserted. The fetched instruction will be executed when STALL– is de-asserted.

All the XL-Series chips must have their STALL– lines tied together.

COND

Condition code input. Goes from the RIP to the RCS. Not normally used outside the processor chip set.

FPCN

Floating point condition code input. This signal goes from the floating point processor to the RCS. In systems without a floating point processor FPCN is tied to ground.

OEAD–

OEAD– is an asynchronous output enable signal for the AD bus. The bus is at a high-impedance state when disabled.

VCC AND GND

VCC is a +5.0 volt supply. GND is a system ground. All VCC and GND pins must be connected—floating pins are not allowed.

NC

No connection (must be left floating). Reserved for future expansion.

TIE HIGH

This signal line is reserved for future expansion. It should be tied to VCC.

TIE LOW

This signal line is reserved for future expansion. It should be tied to GND.

Architecture

BUSES

The XL-8236 uses three buses: the C bus (32 bits), the AC bus (22 bits), and the AD bus (32 bits). The AC (code address) bus is used to address code memory. The AD (data address) bus can be used to transfer data between the RCS registers and the rest of the system, including the XL-8237 RIP, and to save and restore the stack externally. The XL-8237 RIP also uses this bus as a data address bus. The C (code) bus provides the instructions for both the RCS and the RIP.

PIPELINING

The XL-8236 instruction sequence is pipelined. In each clock cycle, the next instruction is fetched while the current instruction is being executed. This parallel fetch/execute architecture allows faster execution than the usual sequential fetch/execute architecture.

INSTRUCTION SET

The instruction set contains branch, conditional branch, subroutine call and return, software interrupt and interrupt return, loop control, and coprocessor control instructions.

INTERRUPT CONTROL

There are three external interrupt lines, plus reset. Interrupts can be masked individually and collectively, with the individual interrupt enable and master interrupt enable bits in the sequencer status register.

Interrupts are vectored to one of fifteen addresses.

MEMORY CONTROL

The RCS's memory interface is controlled by the STALL- input.

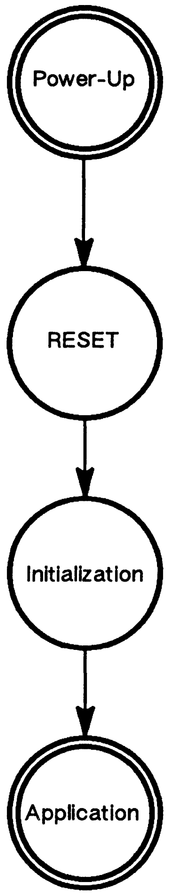
STALL- is used to cancel instructions in the event of delayed code memory word. It is typically used with dynamic RAM and memory caches. External hardware detects memory faults and asserts STALL- until the data is available.

In addition to these two input signals, the RCS has a five-bit OP output bus. The OP bus identifies the current state of the memory interface.

CONTROL FLOW

Figure 3 shows the major states of the system: at power-up, at reset, and in applications programs.

Architecture, continued



At power-up, the state of the XL-8236 is undefined.

When RESET- is asserted, .ibr, .ssr, .cfa, and .cea are initialized, and execution starts from code address 0 (which should contain a continue operation for the RCS, and a no-op for every other processing unit). All other registers are undefined. The initialization of .ssr disables interrupts.

The reset handler should initialize the RCS, the RIP (raster image processor) and any other devices in the system. At this point interrupts should still be disabled. Once the system is initialized, interrupts can be enabled by setting the master interrupt enable bit in the sequencer status register.

Application programs can be called with a branch to subroutine instruction.

Figure 3. Major states of an XL-Series system

Stack

The RCS has a 33-word-deep by 32-bit-wide register file addressed as a stack. The stack may contain loop counts, branch addresses, and subroutine return addresses. The stack consists of a 32-bit top-of-stack register (.tos), a 32-word by 32-bit register file, and a stack pointer, which is a 5-bit field within the *sequencer status register* (.ssr).

On reset, the stack pointer is initialized with all 1's (a value of 31), indicating an empty stack. The stack pointer is a modulo-32 counter which increments before each push and decrements after each pop.

Stack underflow and overflow exceptions are provided. An underflow exception occurs when a pop operation nearly empties the stack. An overflow exception is generated when a push operation nearly fills the stack. (For more details, see sections *Stack Overflow* and *Stack Underflow* on page 15.)

A pair of exception routines can implement a larger stack in system memory. When the RCS stack overflows, it is copied to the main memory stack; when it underflows, data in the memory stack is restored to the RCS stack.

Registers

FETCH AND EXECUTION ADDRESS REGISTERS

The sequencer fetches an instruction on every cycle, and executes it on the following cycle. The instruction cancellation mechanism allows the results of an instruction to be discarded after the instrution has completed, effectively turning the instruction into a no-op, but the chip is never idle.

The currently executing address (.cea) register is a 22-bit register containing the address of the instruction currently being executed. This is the instruction that was fetched on the previous cycle.

The currently fetching address (.cfa) register is a 22-bit register containing the address of the instruction being fetched. This instruction will be executed on the next cycle, and the address in the .cfa will be copied into the currently executing register (.cea).

INTERRUPT ADDRESS REGISTERS

The two interrupt address registers are the interrupt fetch address (.ifa) and the interrupt execute address (.iea). The RCS stores interrupt return addresses in these registers. (For more details see *Interrupt Sequence* on page 10.)

SEQUENCER STATUS REGISTER

The *sequencer status register* (.ssr) is a 32-bit register containing state information. The upper five bits contain the stack pointer. The remaining bits include branch bits, nine sets of flag/enable bits which control and identify the state of interrupts and exceptions, and the master interrupt enable bit. If the *master enable bit* (men) is cleared, all interrupts are prevented from executing.

Several instructions implicitly use or alter information in the .ssr. The .ssr is illustrated in Figures 4 and 5.

The b and bi bits are state bits. The b bit indicates that the previous instruction was a taken branch. The bi bit stores the current values of the b bit for interrupt processing.

The nine sets of flag/enable bits selectively control the interrupt mechanism. If the enable bit is set, and if the indicated exception occurs, an interrupt occurs and the associated flag bit is set. Interrupt-handling software reads this word and examines the flag bits to determine which interrupts have occurred. If either the master or individual interrupt enable is false and an interrupt occurs, no interrupt routine will be called, but the associated exception flag will still be set.

The flag bits are "sticky" — they will remain set even if the signal that sets them goes away. The bits can only be cleared by overwriting the .ssr. If a flag bit is set when its interrupt is disabled, it will *not* cause an interrupt when the interrupt is re-enabled. Thus, all pending interrupts must be handled before exiting the interrupt handler. Furthermore, the interrupts that have been serviced must have their flag bits reset before interrupts are re-enabled (or before the interrupt handler is exited) to assure proper operation of the sequencer.

External interrupts (EXT1–, EXT2–, and EXT4–) must be latched externally until acknowledged.

The five-bit top-of-stack (.tos) pointer is part of the .ssr.

After a reset, the .ssr is initialized with all zeros except for the s bit, which is set; and the .tos field, which is set to all ones to indicate an empty stack.

MODIFYING THE SSR

The .ssr can be read or written as a 32-bit register using the *Intrasystem Data Transfer Instructions* on pages 46–48. While the .ssr can be examined at any time, special care must be exercised when setting it to avoid losing interrupts.

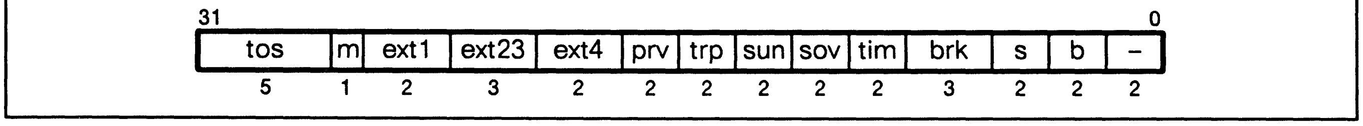


Figure 4. Sequencer status register (.ssr)

Registers, continued

In order to set or reset a portion of the .ssr, a read-modify-write sequence must be performed, that is, the .ssr is read into the RIP, the desired bit manipulation is performed and the .ssr is written with the new value. In order to avoid losing crucial state information and any interrupts that occurred between reading and writing the .ssr, the following rules must be followed:

1. Interrupts should be disabled (that is, the .ssr's men bit should be zero) with a trapi instruction. See pages 16–17.
2. Interrupts from external devices should be held until explicitly acknowledged by the interrupt handler software.
3. No internal interrupt, except timer interrupts, should occur.

TIMER REGISTER AND INTERRUPT

The RCS includes a programmable timer based on a 32-bit timer register. The timer register contains a signed 32-bit number whose value represents the number of clock cycles remaining until a timer interrupt occurs. The timer register is decremented during every clock cycle. Whenever the value is negative, the timer flag is set, allowing a timer interrupt occur.

The timer will continue to decrement even when negative. This allows accurate timing if the service routine is interrupted or delayed.

BREAKPOINT REGISTER

The XL-8236 also includes a 32-bit breakpoint register (.brk), used to provide a code breakpoint for program development. A code address can be loaded into the .brk register. If the .ssr brkenc bit is set, any attempt to fetch the instruction located at the address loaded in the .brk register sets its breakpoint interrupt flag, generating a code-break interrupt if enabled.

Symbol	Bit #	Meaning	
-	0	-	reserved: must be set to zero
	1	-	reserved: must be set to zero
b	2	b	last instruction was a taken branch
	3	bi	last instruction of interrupted process was a taken branch
s	4	s	reserved: must be set to one
	5	si	reserved: must be set to one
brk	6	brkflg	flag for breakpoint interrupt
	7	brkenc	enable for code breakpoint interrupt
	8	-	reserved: must be set to zero
tim	9	timflg	flag for timer interrupt
	10	timen	enable for timer interrupt
sov	11	sovflg	flag for RCS stack overflow interrupt
	12	soven	enable for RCS stack overflow interrupt
sun	13	sunflg	flag for RCS stack underflow interrupt
	14	sunen	enable for RCS stack underflow interrupt
trp	15	trpflg	flag for trap instruction interrupt
	16	trpen	enable for trap instruction interrupt
prv	17	prvflg	reserved: must be set to zero
	18	-	reserved: must be set to zero
ext4	19	ext4flg	flag for external interrupt 4
	20	ext4en	enable for external interrupt 4
ext23	21	ext23en	enable for external interrupt 2
	22	ext2flg	flag for external interrupt 2
	23	-	reserved: must be set to zero
ext1	24	ext1flg	flag for external interrupt 1
	25	ext1en	enable for external interrupt 1
men	26	men	master interrupt enable
.tos	31-27	.tos	five-bit top-of-stack pointer

Figure 5. Bit fields in the sequencer status register

Neutralization

The XL-8236 RCS and its companion, the XL-8237 RIP, achieve high speed by simultaneously fetching the next instruction while executing the current instruction. When a branch is executed, the RCS already has the instruction following the branch in its instruction pipeline. This instruction is called the "shadow instruction." Fetching the instruction at the branch address takes an additional cycle, since it's not yet in the pipeline, so the destination instruction is executed after a one-cycle delay. This is called "delayed branching".

The XL-8236 provides a neutralization output line, NEUT–. It can selectively cancel the effects of the shadow instruction, effectively replacing it with a no-op. The XL-8236 instruction set normally sets NEUT– active after branch, call and return instructions (including interrupt calls and returns), thereby canceling the shadow instruction. This allows the programmer to ignore the effects of delayed branching.

Neutralized instructions actually run to completion, but their results are discarded at the end of the cycle. Registers, status flags, and so on are simply not updated, so internal effect is as if the instruction was never executed.

The XL-8236 instruction set also provides three additional instructions which allow the shadow instruction to be executed: override neutralization (ovneut), override and increment stack pointer (ovneuti), and reverse neutralization (revneut). Efficient code makes use of these instructions to selectively execute shadow instructions, saving up to one clock cycle per branch.

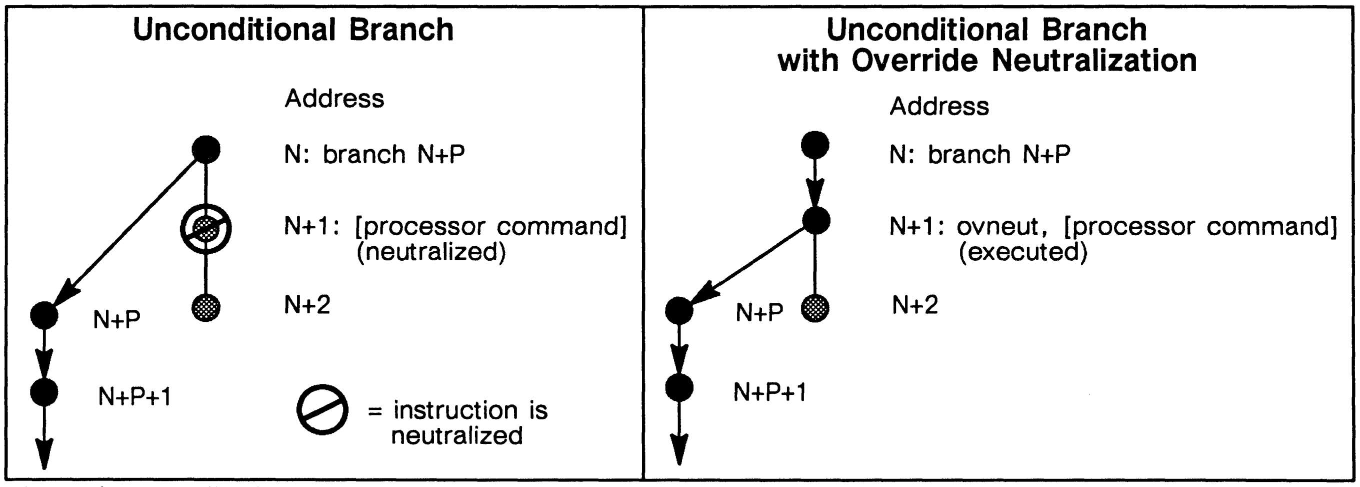

Unconditional Branch
Address
N: branch N+P
N+1: [processor command] (neutralized)
N+P
N+2
N+P+1
= instruction is neutralized
Unconditional Branch with Override Neutralization
Address
N: branch N+P
N+1: ovneut, [processor command] (executed)
N+P
N+2
N+P+1

Figure 6. Neutralization

STALL–

The XL-8236 has a control input, STALL–, which directs the RCS and RIP to cancel the effects of the next instruction. This signal is asserted by external hardware when unable to complete a bus transfer in time, such as during a cache miss or refresh cycle. An active STALL– signal cancels the next instruction. Instructions canceled through this mechanism are refetched and will be re-executed when the STALL– signal is de-asserted.

Cancelled instructions actually run to completion, but their results are discarded at the end of the cycle. Registers, status flags, and so on are simply not updated, so internal effect is as if the instruction was never executed.

STALL– allows the current instruction to complete, but cancels the next instruction. The most common use of STALL– is to re-execute an instruction fetch on a wait state, cache miss, or refresh cycle. The invalid word loaded on the STALL–ed cycle is released, then the instruction is executed and normal execution continues.

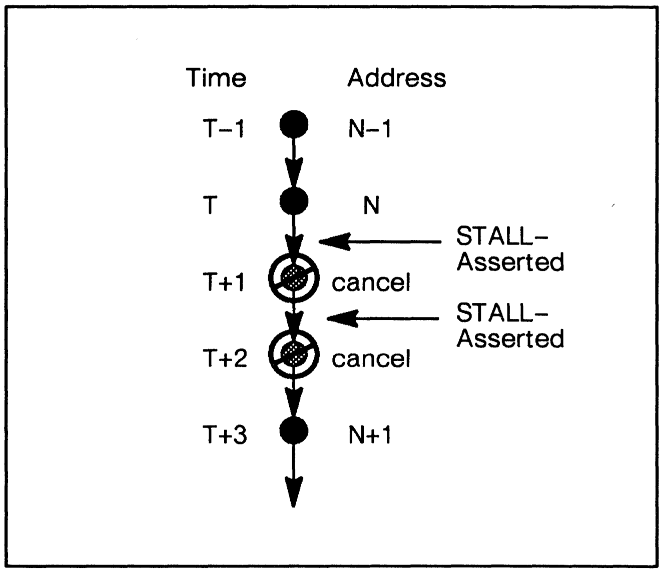


Figure 7. Effects of STALL–

Interrupts

The XL-8236 can receive interrupts from three external sources: EXT1–, EXT2–, and EXT4– (there is no EXT3); and can generate five interrupts internally: BRK, TIM, SOV, SUN, and TRP. When an interrupt control line or internal condition becomes active, the RCS sets the corresponding .ssr interrupt flag. If the master interrupt enable (men) bit of the .ssr is set, and the corresponding .ssr interrupt enable is active, the interrupt will be honored, as described below.

There are fifteen interrupt vector addresses. All external interrupt lines are level-sensitive. They are sampled at the rising edge of the clock.

INTERRUPT SEQUENCE

When an interrupt is detected, the .cfa is stored in the iea, and the next fetch address is placed in the .ifa. This sequence allows the system to return to the next instruction (.cfa) on an interrupt return. See figure 8.

The RCS then neutralizes the fetched instruction and branches to the interrupt vector address. The old value of the .ssr b bit is saved in the bi bit.

The interrupt vectoring scheme is based on four classes of interrupts. When an interrupt request is approved, the RCS branches to the address formed by or-ing the 32-bit interrupt base address register (.ibr) with the four interrupt classes as shown in figure 12. This gives the capability of up to 15 different vector addresses (not 16 because at least one class bit must be non-zero for an interrupt to occur).

Note that if multiple interrupts occur simultaneously, they are not prioritized. Rather, the vector address of the interrupt handler is selected to indicate which interrupt classes are pending.

Interrupts can be nested to any depth by saving the contents of the .iea, .ifa and .ssr registers externally.

RETURNING FROM INTERRUPTS

To return from an interrupt, two special interrupt return instructions must be executed, return-from-interrupt-0 (rfi0) and return-from-interrupt-1 (rfi1). Executing rfi0 returns the .iea register to the .cfa register places the contents of the .iea register onto the AC bus and enables the interrupt master enable bit (men). The upadating of the men bit may occur on the cycle in which the rfi0 is executed, or one cycle after that. Executing rfi1 returns the .ifa to the .cfa and places the .ifa register contents onto the AC bus, restoring the RCS state to what it was before the interrupt was requested.

Interrupts, continued

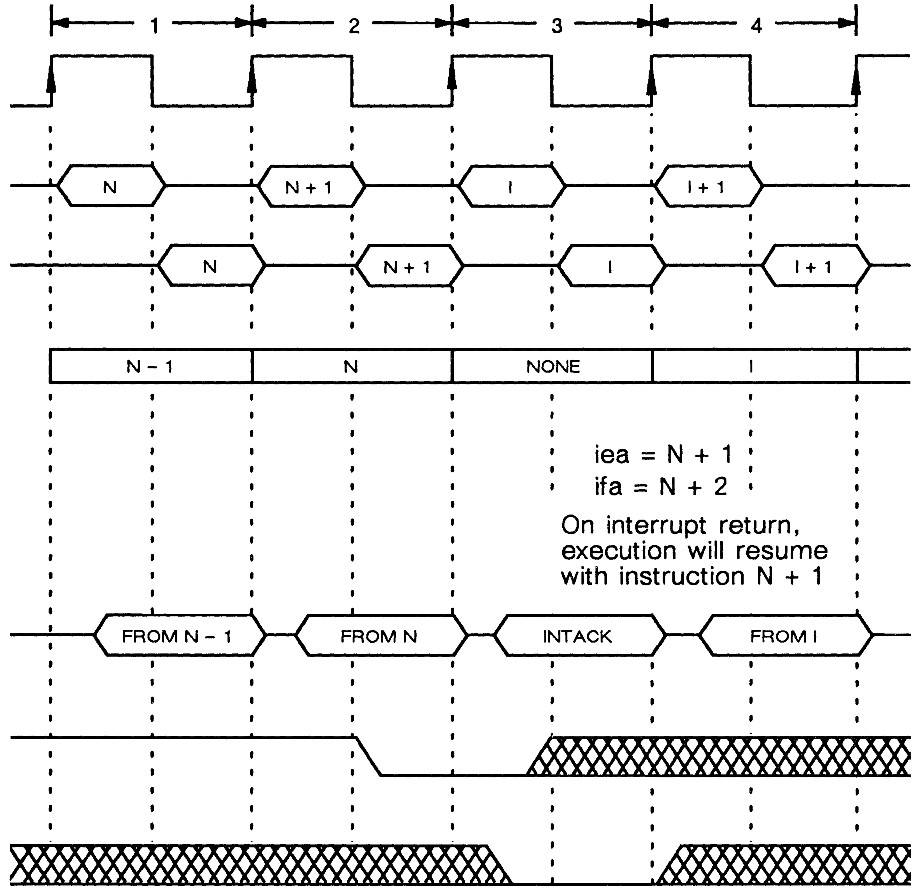


Figure 8. Normal interrupt entry sequence

Note that the enabling of the men bit during the rf10 cycle allows an interrupt to be acknowledged before the rfi1 instruction executes, causing the state of the machine to be lost. RfiO also ignores the state of STALL–. Two steps must be taken to avoid this from causing trouble:

4. Condition C13 by ANDing it with STALL–, as shown in figure 9. This will prevent the RCS from ever seeing an rfiO instruction during STALL– cycles, and eliminates the problem (that it alters the code word is unimporant, since the STALL– signal will cause it to not be executed). If your design does not use STALL–, you do not need to implement this.

5. Use the code in figure 10 at the start of your interrupt routine to test for improper interrupt exit sequences and to restore the state correctly. This step must be implemented whether you use STALL– or not.

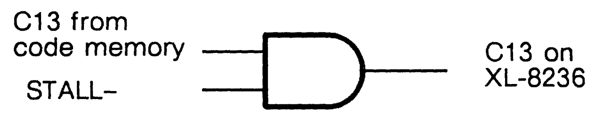


Figure 9. Hardware portion of the special handling for rfi0/rfi1

Interrupts, continued

```
/* Beginning of interrupt handler */
/*      This routine works by testing the value of .iea against the address of the interrupt
        handler's rfi1 instruction. This implementation assumes that there is only one exit routine;
        that is, that there is only one rfi1 instruction in the whole system.

        The extra overhead consists only of a few instructions, since the interrupt calls are not
        spurious; they simply happened a cycle too soon.

        This example is written in a pseudo-code that mixes C and XL assembly code. */
swap register banks
if (.iea == rfi1_addr) {
        /* This interrupt came between rfi0 and rfi1. Restore the state of the previous call, with
           a few exceptions... */
        saved_ssr.bi = .ssr.bi;
} else {
        /* This is a normal interrupt call. Save .iea, .ifa, and .ssr specially */
        saved_iea = .iea;
        saved_ifa = .ifa;
        saved_ssr = .ssr;
}

/* Main part of interrupt handler */
...
/* End of interrupt handler */
...
.iea = saved_iea;
.ifa = saved_ifa;
.ssr = saved_ssr;

rfi0;
rfi1_addr:
rfi1; asrtadr
```

Figure 10. Software portion of the special handling of rfi0 and rfi1

INTERRUPTS AND STALL–

Interrupt processing takes precedence over stalls: if STALL– is asserted and an enabled interrupt is approved, the RCS will honor the interrupt and perform the interrupt entry sequence. (See figure 11). Because the AC bus address changes (the only situation when it can change with STALL– asserted), designers using variable-latency code memory subsystems must handle this case. This is described in detail in the *XL-Series Hardware Designer's Guide.*

Interrupts, continued

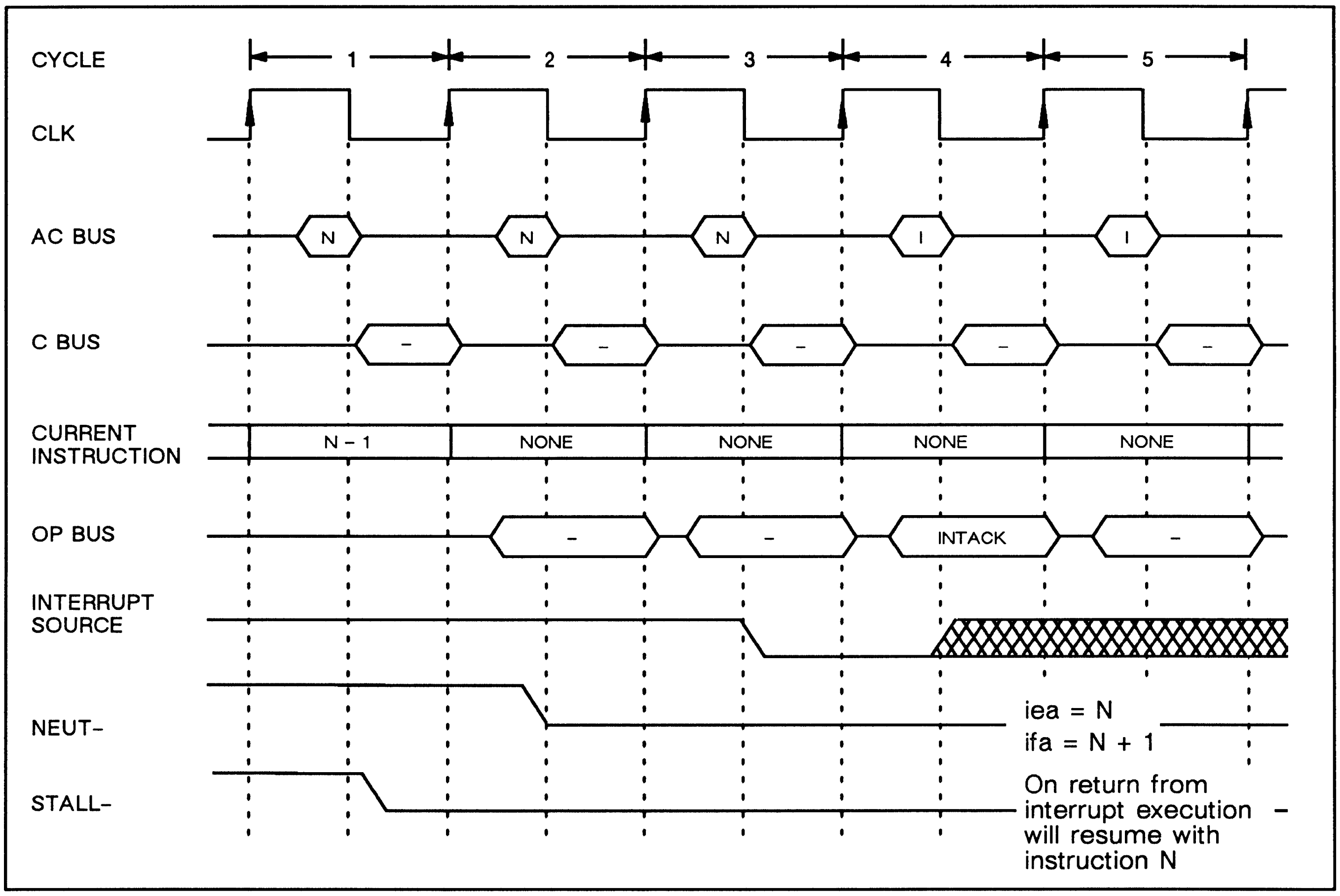


Figure 11. Interrupt with STALL- asserted

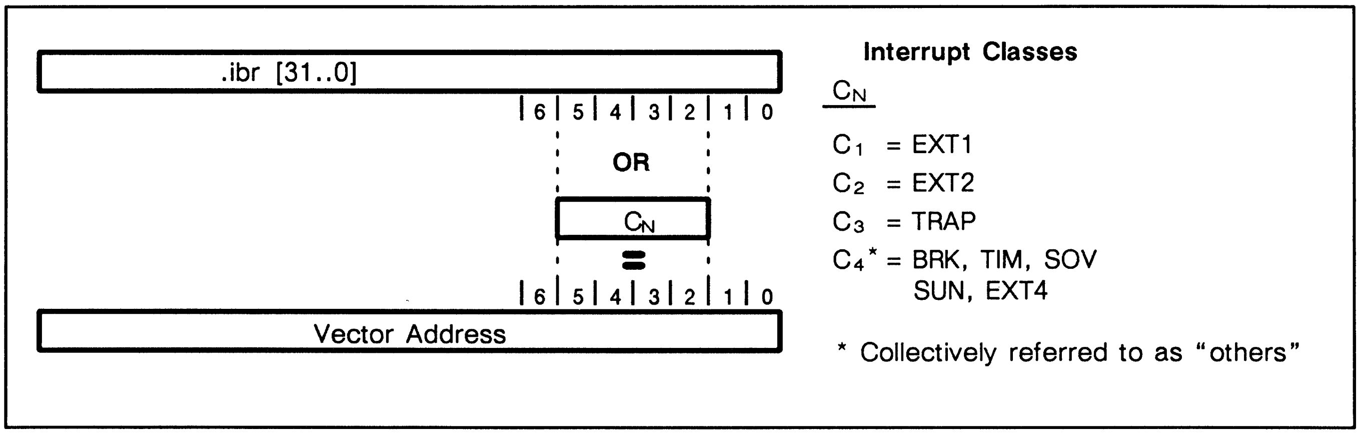


Figure 12. Interrupt vector address

INTERRUPT FLAG BITS

The interrupt flag bits record the history of the associated interrupt. If the interrupt was asserted at any time in the past, then the flag bit will be set even if the enable is not set (except for the breakpoint register; breakpoint comparisons are only performed if the associated enable bit is set). Once the flag bit is set it can only be cleared by writing an entire word into the .ssr.

Interrupts, continued

EXTERNAL INTERRUPT SOURCES

The external interrupt sources are: EXT1–, EXT2–, and EXT4–. Each has status and interrupt enable bits in the .ssr.

The EXT1– control line is a dedicated external interrupt. Its interrupt mask bit in the .ssr is ext1en. Its status bit is ext1flg.

The EXT2– interrupt line has an enable bit, ext23en. Its status bit is ext2flg.

EXT4– is typically used to signal exception conditions from floating point processors, but can be used as a general-purpose interrupt. Its enable bit is ext4en, and its status flag is ext4flg.

INTERNAL INTERRUPT SOURCES

The five internal interrupt sources (SOV, SUN, TRP, TIM, and BRK) each have a status and interrupt enable bit in the .ssr.

SOV and SUN indicate stack near-overflow and near-underflow. SOV occurs when data is pushed into the third-to-last available word on the stack (.tos = 29). SUN occurs when the stack is empty or nearly empty (.tos = 1 or .tos = 0, or .tos = 31). The enable and status bits for SOV and SUN are soven, sovflg, sunen, and sunflg, respectively.

Note that the stack underflow exception can occur at more than one stack position. The stack pointer (the .tos field) must be used to determine the stack position in exception handling, rather than using constants showing the stack position at which you expect the exception to occur.

TRP is set by invoking the trap instruction. Its enable and status bits are trpen and trpflg, respectively.

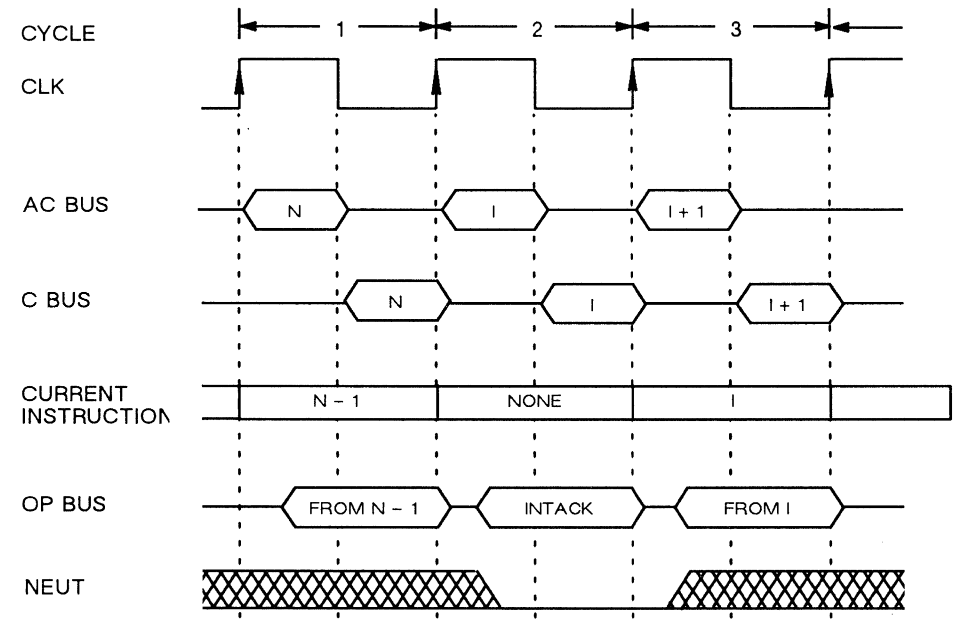
The remaining two exceptions, TIM and BRK, are set on timer interrupts and breakpoints, respectively. Their enable and status bits are timen, timflg, brkenc, and brkflg.

BREAKPOINT FACILITY

The breakpoint (.brk) register provides a facility to interrupt normal program execution when a specific instruction is executed (breakpoint).

See figure 13 for the timing of a code breakpoint. The system stops *before* executing the instruction referenced by the .brk register. Note that even instructions which are to be neutralized will cause a code breakpoint. This allows simple single-stepping of the system by setting the breakpoint to the .ifa register (the "next" instruction to be executed).

Breakpoints are not reliable if they occur on addresses that are executed *after* shadow instructions, such as branch targets.



iea = N
ifa = N + 1

The breakpoint register contains "N." On return from interrupt, execution will resume from instruction "N."

Figure 13. Code breakpoint timing

Interrupts, continued

STACK OVERFLOW

The sovflg bit of the .ssr is set when the .tos field contains a 29 and a stack push operation is performed. If the stack overflow interrupt is enabled (soven = 1 and men = 1), then it will be detected during the cycle after the completion of the push operation. See figure 14 for details.

STACK UNDERFLOW

The sunflg bit of the .ssr is set when the .tos field contains a 1, or 0, or 31 and a stack pop operation is performed. If the stack underflow interrupt is enabled (sunen = 1 and men = 1), then it will be detected either one or two cycles after the completion of the pop operation. (The extra cycle of delay doesn't cause problems because the exception is triggered when the next-to-last word is popped off the stack. Even if another pop occurs in the cycle between the first pop and the assertion of the exception, the data popped off will still be valid.) See figure 14 for details.

The stack should be initialized after a reset by pushing two values onto the stack. This will move it past the point where the stack underflow interrupt occurs.

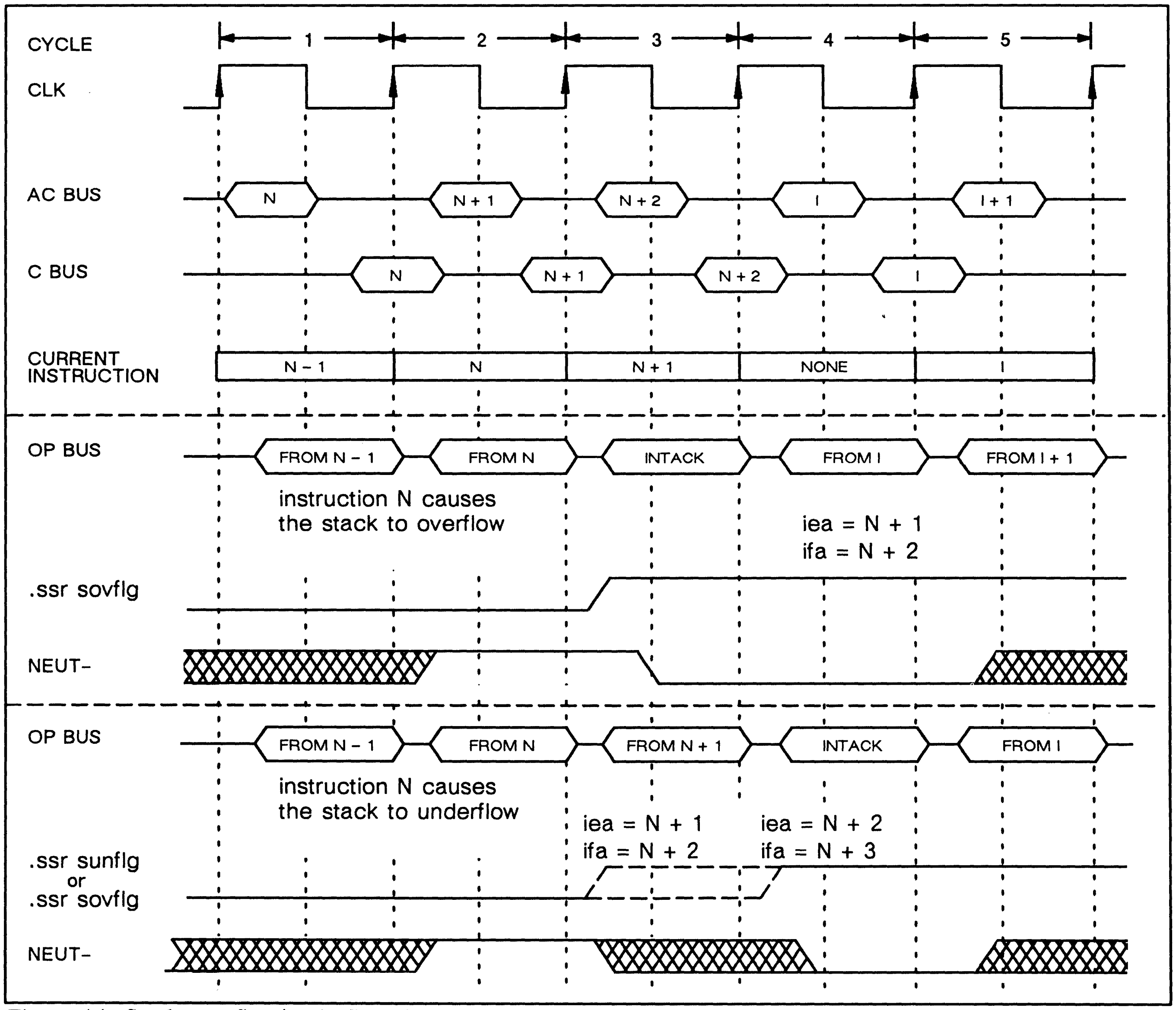


Figure 14. Stack overflow/underflow timing. Note that the underflow may be signaled on one of two cycles.

Interrupts, continued

TRAPS

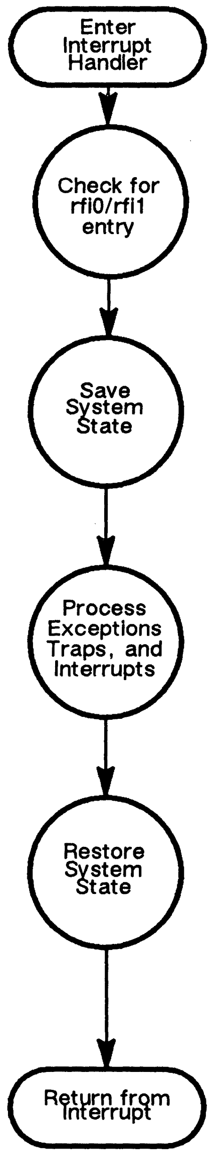
Software interrupts on the XL-8236 are called *traps*. They are invoked with the trapi instruction.

Traps are used primarily for system calls. The programmer would specify a system call by using the immediate field of the trap instructions, as in "trapi 47." This would push the number 47 onto the stack and cause a trap interrupt. The trap handler would use the value on the stack as a parameter.

HANDLING INTERRUPTS

The software that handles interrupts should follow the procedure given in figure 15. The handler for nested interrupts is more complex. This is shown in figure 16 (detail in steps common to both figures is skimpy in figure 16, so be sure to read both).

Each flag bit in the .ssr must be reset after the interrupt or exception is serviced.



When an interrupt or exception occurs, the RCS disables interrupts (by clearing the men bit in the .ssr) and branches to one of the fifteen interrupt vectors. For a simple interrupt handler, all fifteen vectors can contain a branch to the same interrupt handling code.

Check to see if the handler was entered between an rfi0 and rfi1 instruction. Fix up the return address if so.

All registers that could be modified by the interrupt handler must be saved. Latencies must be taken into account; the multiply/divide unit on the RIP may be in the middle of a divide operation, which takes 16 cycles. The FPU, if present, may also take a number of cycles to complete an operation. If the RIP registers (.r0-.r31) are saved first, all other registers will be in a stable state by the time they are examined.

The floating point condition on the XL-3232 is not part of the status register. You can test it by performing a floating point branch and testing whether the branch was taken or not.

Internal exceptions are not "sticky;" that is, they do not re-assert themselves if they aren't processed. If you return from an interrupt without processing an exception, the interrupt handler will not be re-entered. If the unprocessed exception affects the application program (for example, if it was a stack overflow and the application program does a push), bad things will happen. Thus, you should process all asserted exceptions during every call to the interrupt handler. You should also test for floating point exceptions on every call to the interrupt handler, if the system has a floating point unit.

You do not have to handle all *external* interrupts on every call to the interrupt handler, since they are required to be latched externally, and thus will be re-asserted when interrupts are re-enabled.

All registers are restored in a straightforward way except .ssr (and possibly .tim). For .ssr, the old value of the bi bit must be restored. The current value of .tos must be maintained. The interrupt and exception enables will usually be restored to their old state.

The timer register, .tim, can be restored if you are using it to measure "user" time, or left as it is if you are measuring elapsed time.

On the XL-3232, the floating point condition can be set by performing a dummy floating point operation.

To return from interrupt, execute this code:

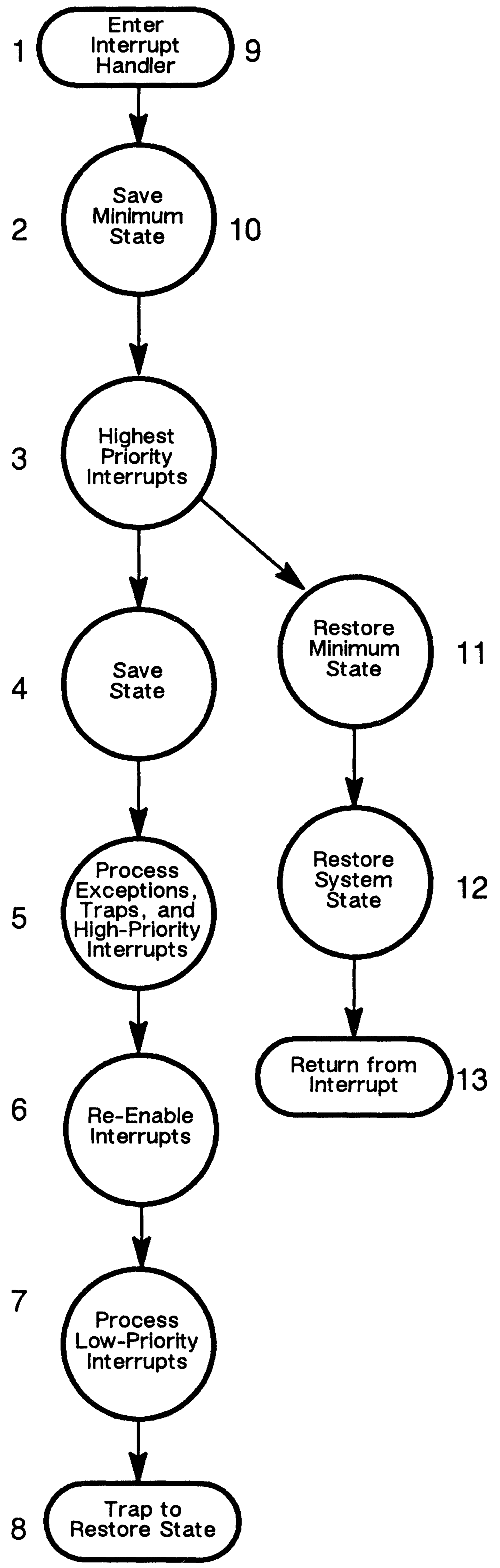
```
rfi0
rfi1; asrtadr
```

The asrtadr instruction is a RIP instruction that forces the .adr register onto the AD bus, which restarts any interrupted address-generation instructions.

The rfi0/rfi1 instruction sequence re-enables interrupts by setting the men bit to its previous state.

Figure 15. Description of interrupt handling

Interrupts, continued



1. The only way to turn interrupts off reliably is with a trap instruction, since masking the master enable bit with a "mov ra, .ssr" instruction is itself interruptable. You need to disable interrupts before you restore state. Thus, you should assign a value of the immediate field of a trap instruction to mean "this is a request to restore state."
2. "Minimum state" consists of those registers that may be changed when we test for the restore-state request. To test the immediate value on the stack you must copy it to the RIP and compare it to a constant, forcing you to overwrite at least two RIP registers and the processor status register. These registers, and any other registers modified by these tests, must be saved first.
3. Very high priority interrupts can also be processed here, but extreme care must be taken to avoid stack overflow and the overwriting of registers that aren't saved with the minimum state.
4. Save the rest of the state of the machine, as in the previous example. State must be saved on a stack for nested interrupts, to allow the state to be restored correctly as the interrupts un-nest.
5. In addition to traps and exceptions, the high-priority external interrupts should be processed first before interrupts are turned back on. Stack overflow should be processed first to allow some level of subroutine nesting.
6. Interrupts can be re-enabled by writing the .ssr to the RIP, masking the master enable bit on, then copying it back to the .ssr.
7. Low-priority interrupts are processed next. If another interrupt occurs during processing, execution will proceed from step 1.
8. This consists of a trapi instruction. For example, if 42 has been chosen as the code for "restore state," this step consists of a "trapi 42" instruction.
9. The interrupt handler is re-entered after the trapi instruction.
10. Minimum state is saved, and, if there is a trap pending, the immediate value on the stack is examined for the "restore state" code.
11. Since the request was "restore state," the minimum state is restored...
12. Then the rest of the state is restored...
13. Then the interrupt handler is exited with the usual

```
            rfi0
            rfi1; asrtadr
```

    sequence.

Figure 16. Nested interrupts

RESET

Activating the RESET– line at the end of the clock cycle initializes the .ssr according to Figure 17, sets the .cfa register to zeros and sends the value zero out on the AC bus, forcing a branch to address zero. Figure 18 shows detailed reset timing. The user should place a no-op instruction in location zero.

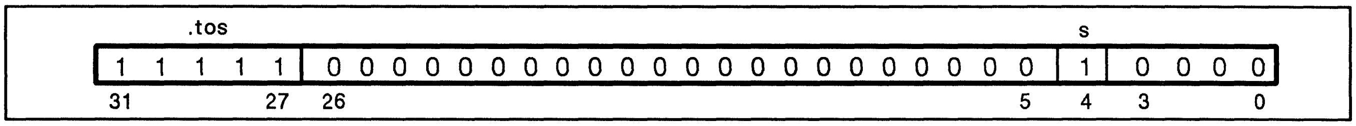


Figure 17. Sequencer status register, initialized

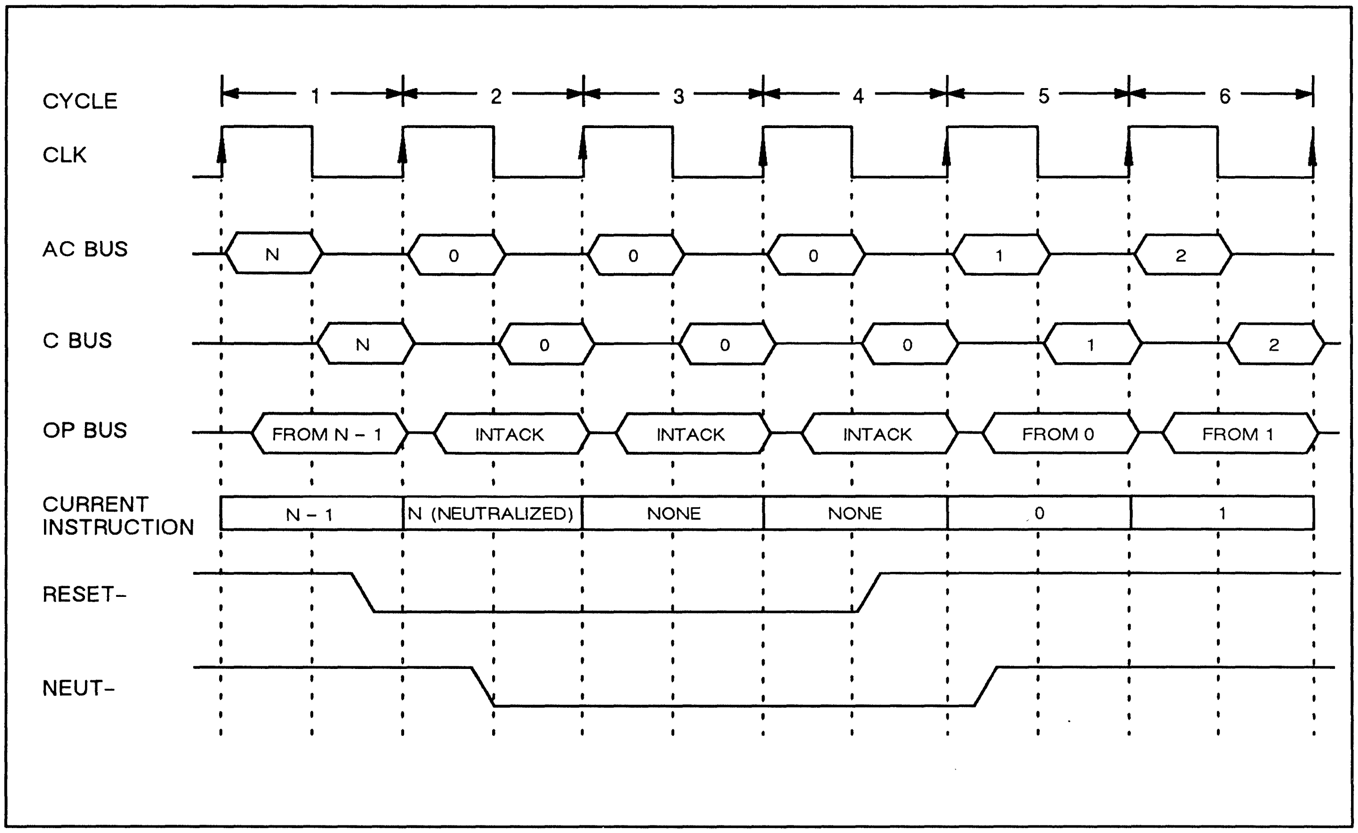


Figure 18. Reset timing

Instruction Set

TERMS AND SYMBOLS

The instructions are listed on pages 20 and 21, then described in detail on the following pages. Each description includes a pseudo-code definition of the instruction. The following symbols are used:

||	Concatenate fields. abc || def gives abcdef.	COND	Condition Code
|	Indicates that operations separated by this symbol occur in parallel.	{ }	Begin and end comment
a dup b	Duplicate b a times. 3 dup 0 gives 000.	% ixs	Shift left by ixs bits
stack(t)	Location t in the sequencer stack	[31..0]	Specifies the bit field from bit 31 to bit 0, inclusive. For example, reg (ra) [3..0] gives the lower four bits of register ra.

Figure 19. Terms and symbols

INSTRUCTION FORMAT

The XL-8236 uses two instruction formats: short and long. Short instructions use the upper 8 bits of a 32-bit instruction; the remaining bits are used as the instruction field for the XL-8237 RIP or a coprocessor. Short instructions include neutralization control, short branches with a 5-bit displacement and branching from a 32-bit displacement on the stack.

Long instructions are 32 bits long, with a format recognized by the RIP as a RCS operation. These instructions are used to provide bus transfer and housekeeping control operations, in addition to 24-bit and 28-bit subroutine and branch immediates.

Coprocessor instructions are reserved for future expansion, with the exception of the coprocessor load/store instructions, which are used in the XL-8232 to load and store the graphics floating point unit.

Field	Meaning
ra	selects ra register (of processor)
imm11, imm24, imm28	11, 24, or 28 bit immediate
c	condition polarity select
imm5	5-bit signed or unsigned immediate
ext1, ext2	operation code extensions
rd	selects rd register (of processor or coprocessor)
extn	external register number n

Figure 20. Instruction fields

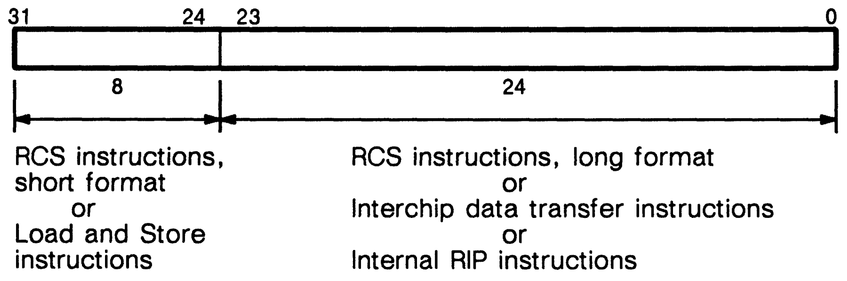

31
24
23
0
8
24
RCS instructions, short format
or
Load and Store instructions
RCS instructions, long format
or
Interchip data transfer instructions
or
Internal RIP instructions

Figure 21. Instruction formats

Instruction Format, continued

	Format	Detailed Description Page #
Continue		
Cont	[000][00110][RIP/coprocessor]	22
Branch		
Branch	[000][00010][imm24]	24
Short branch	[100][imm5][RIP/coprocessor]	25
Branch to stack and pop	[000][01110][RIP/coprocessor]	26
Short forward branch on condition	[01][c][imm5][RIP/coprocessor]	27
Loop Control		
Loop enter	[000][00111][RIP/coprocessor]	29
Absolute branch to stack or or pop on condition	[000][01][c][00][RIP/coprocessor]	30
Decrement stack and branch or pop if zero	[000][00001][imm24]	31
Decrement stack and backward branch or pop if zero	[001][imm5][RIP/coprocessor]	32
Branch and pop (loop exit)	[000][00011][imm24]	33
Subroutine Control		
Subroutine call	[0001][imm28]	35
Subroutine return	[000][01101][RIP/coprocessor]	36
Interrupt Control		
Return from interrupt 0	[00000000][x][00101][x]	38
Return from interrupt 1	[000][00100][RIP/coprocessor]	39
Trap		
Trap immediate	[00000000][11][x][00010][imm11]	40
Neutralization		
Override neutralization	[000][01011][RIP/coprocessor]	43
Override neutralization of subroutine call shadow	[000][01111][RIP/coprocessor]	44
Reverse neutralization	[000][00101][RIP/coprocessor]	45

Instruction Format, continued

Intrasystem Data Transfer Instructions

									Detailed Description Page #
Transfer word from RIP to RCS internal register	000	00000	001		ra		ext2	x	46
	3	5	3		5		5	11	
Transfer word from Coprocessor to RCS internal register	000	00000	11		x		ext2	x	
	3	5	2		6		5	11	
Transfer word from RCS internal register to RIP	000	00000	011		ra		ext2	0	48
	3	5	3		5		5	11	
Transfer word from RCS internal register to Coprocessor	000	00000	11		x		ext2	x	
	3	5	2		6		5	11	
Pop stack to RIP register	000	00000	010		ra		00000	x	50
	3	5	3		5		5	11	
Pop stack to Coprocessor	000	00000	101		x		00000	x	
	3	5	3		5		5	11	
Copy stack to RIP register	000	00000	010		ra		00001	x	51
	3	5	3		5		5	11	
Copy stack to Coprocessor	000	00000	101		x		00001	x	
	3	5	3		5		5	11	
Push RIP register onto stack	000	00000	000		ra		00000	x	52
	3	5	3		5		5	11	
Push Coprocessor register onto stack	000	00000	100		x		00000	x	
	3	5	3		5		5	11	
Transfer word from RIP register to external register	000	00000	001		ra	1	extn	x	53
	3	5	3		5		4	11	
Transfer Coprocessor register to external register	000	00000	11		x	1	extn	x	
	3	5	2		6		4	11	
Transfer word from external register to RIP	000	00000	011		ra	1	extn	x	54
	3	5	3		5		4	11	
Transfer external register to Coprocessor register	000	00000	11		x	1	extn	x	
	3	5	2		6		4	11	
Store Coprocessor	101	rd	RIP/coprocessor						55
Load RIP	110	rd	RIP/coprocessor						
Load Coprocessor	111	rd	RIP/coprocessor						
Store RIP	000	01001	RIP/coprocessor						
	3	5	24						

Continue Instruction

CONTINUE

000	00110	RIP/coprocessor
3	5	24

FORMAT

cont

DESCRIPTION

This instruction causes instruction fetching to proceed in normal, sequential fashion. The lower 24 bits are used for either an XL-8237 instruction or a coprocessor instruction. Short instruction format.

NEUTRALIZATION

This instruction is neutralized if the b field of the .ssr contains a one.

OPERATION

```
if b = 1 then
      NEUT := true;
      b := 0;
else
      NEUT := false;
endif;
cfa := cfa + 1;
AC := cfa;
```

Branch Instructions Summary

The following instructions are used for branching:

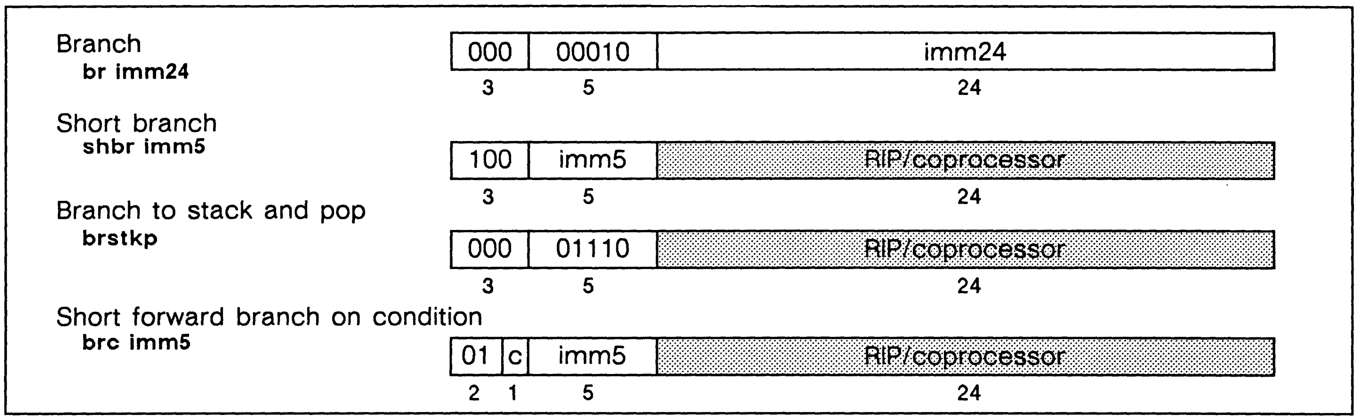


Figure 22. Format of branch instructions

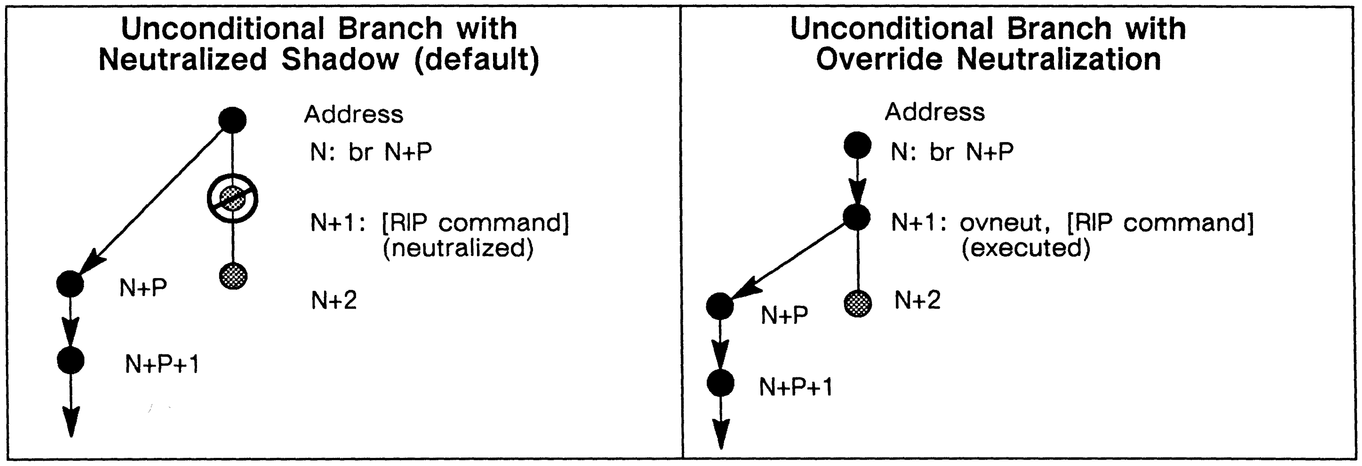


Figure 23. Unconditional branch timing. Valid for br imm24, shbr imm5, and brstkp.

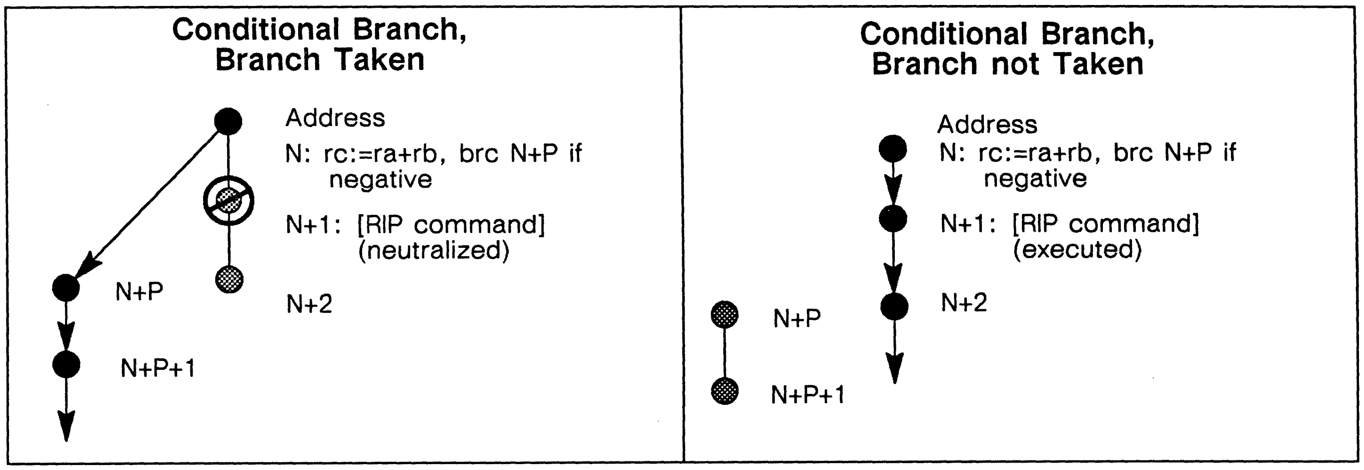


Figure 24. Conditional branch timing—brc imm5 instruction.

Branch Instructions, continued

BRANCH

000	00010	imm24
3	5	24

FORMAT

br imm24

DESCRIPTION

Causes the RCS to branch to the address specified by adding the sign-extended 24-bit immediate imm24 to the address of the currently executing instruction. Long instruction format.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one. If the branch is taken, the b bit of the .ssr is set to one.

EXCEPTIONS

None

OPERATION

```
if b = 1 then
    NEUT := true;
    cfa := cfa + 1;
    b := 0;
else
    NEUT := false;
    cfa := cea + (9 dup imm24[23]) || imm24[22..0];
    b := 1;
endif;
AC := cfa;
```

Branch Instructions, continued

SHORT BRANCH

100	imm5	RIP/coprocessor
3	5	24

FORMAT

shbr imm5

DESCRIPTION

Causes a branch to the address specified by adding the sign-extended 5-bit offset to the currently executing instruction. Short instruction format.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one. If the branch is taken, the b bit of the .ssr is set to one.

EXCEPTIONS

None

OPERATION

```
if b = 1 then
      NEUT := true;
      cfa := cfa + 1;
      b := 0;
else
      NEUT := false;
      cfa := cea + (28 dup imm[4]) || imm5[3..0];
      b := 1;
endif;
AC := cfa;
```

Branch Instructions, continued

BRANCH TO STACK AND POP

000	01110	RIP/coprocessor
3	5	24

FORMAT

brstkp

DESCRIPTION

This instruction takes a branch to the address specified by adding the value on the top of stack to the address of the currently executing instruction. The top of stack value is popped off. Short instruction format.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one. The b bit of the .ssr is set to one.

EXCEPTIONS

Stack underflow

OPERATION

```
if b = 1 then
        NEUT := true;
        cfa := cfa + 1;
        b := 0;
else
        NEUT := false;
        cfa := cea + stack(tos);
        tos := tos - 1;
        b := 1;
endif;
AC := cfa;
```

Branch Instructions, continued

SHORT FORWARD BRANCH ON CONDITION

01	c	imm5	RIP/coprocessor
2	1	5	24

FORMAT

brc imm5

DESCRIPTION

If the selected condition is active, causes a branch to the address specified by adding the zero-extended 5-bit offset to the address of the currently executing instruction. If the 24-bit processor/coprocessor field contains a coprocessor instruction, then FPCN is tested; otherwise, COND is tested. See the *XL-8237 Data Sheet* for details of the processor/coprocessor instruction field. Short instruction format.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one. If the branch is taken, the b bit of the .ssr is set to one.

EXCEPTIONS

None

OPERATION

```
if b = 1 then
      NEUT := true;
      cfa := cfa + 1;
      b := 0;
else
      NEUT := false;
      if {processor operation is coprocessor} then
            cond := FPCN;
      else
            cond := COND;
      endif;
      if c · cond then
            cfa := cfa + 1;
      else
            cfa := cea + (27 dup 0) || imm5[4..0];
            b := 1;
      endif;
endif;
AC := cfa;
```

Loop Control Instructions Summary

The following instructions are used for loop control:

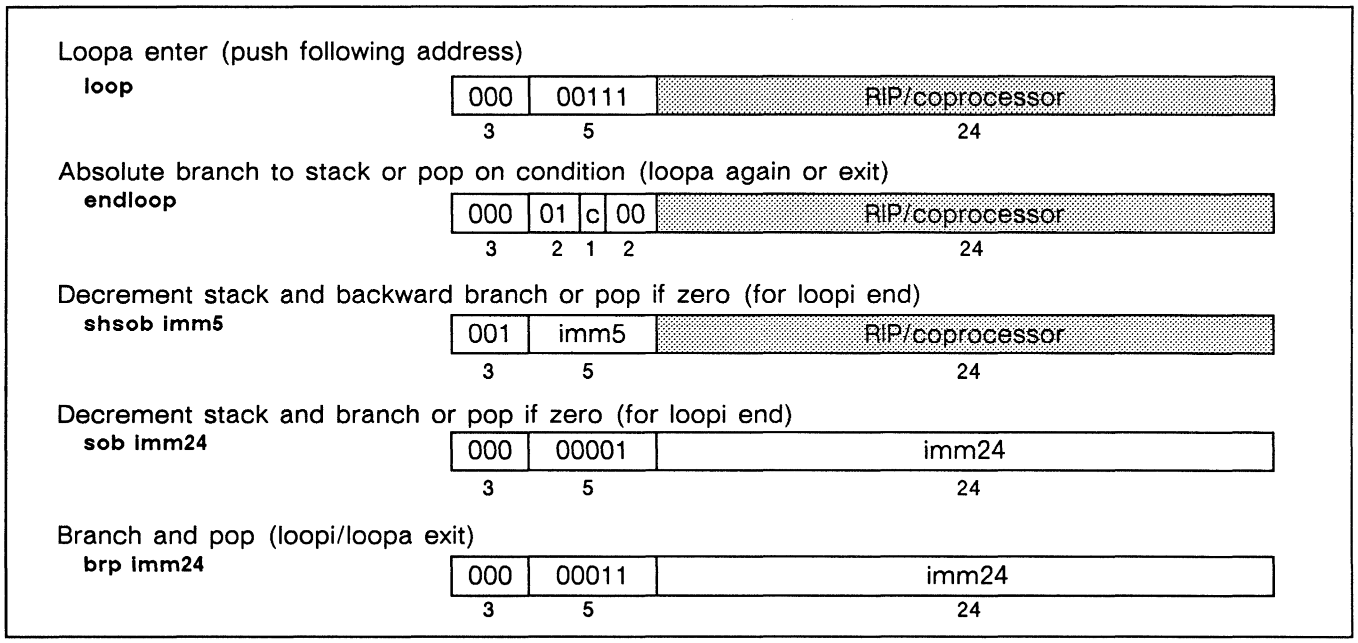

Loopa enter (push following address)
loop
000 | 00111 | RIP/coprocessor
3 | 5 | 24
Absolute branch to stack or pop on condition (loopa again or exit)
endloop
000 | 01 | c | 00 | RIP/coprocessor
3 | 2 | 1 | 2 | 24
Decrement stack and backward branch or pop if zero (for loopi end)
shsob imm5
001 | imm5 | RIP/coprocessor
3 | 5 | 24
Decrement stack and branch or pop if zero (for loopi end)
sob imm24
000 | 00001 | imm24
3 | 5 | 24
Branch and pop (loopi/loopa exit)
brp imm24
000 | 00011 | imm24
3 | 5 | 24

Figure 25. Loop control instruction format

There are two kinds of loops: loops that run for a fixed number of iterations, using a loop count on the stack; and loops that run until a condition is met, with the top-of-loop branch address on the top of stack. The fixed-count loop is called a loopi; the conditional loop is called a loopa.

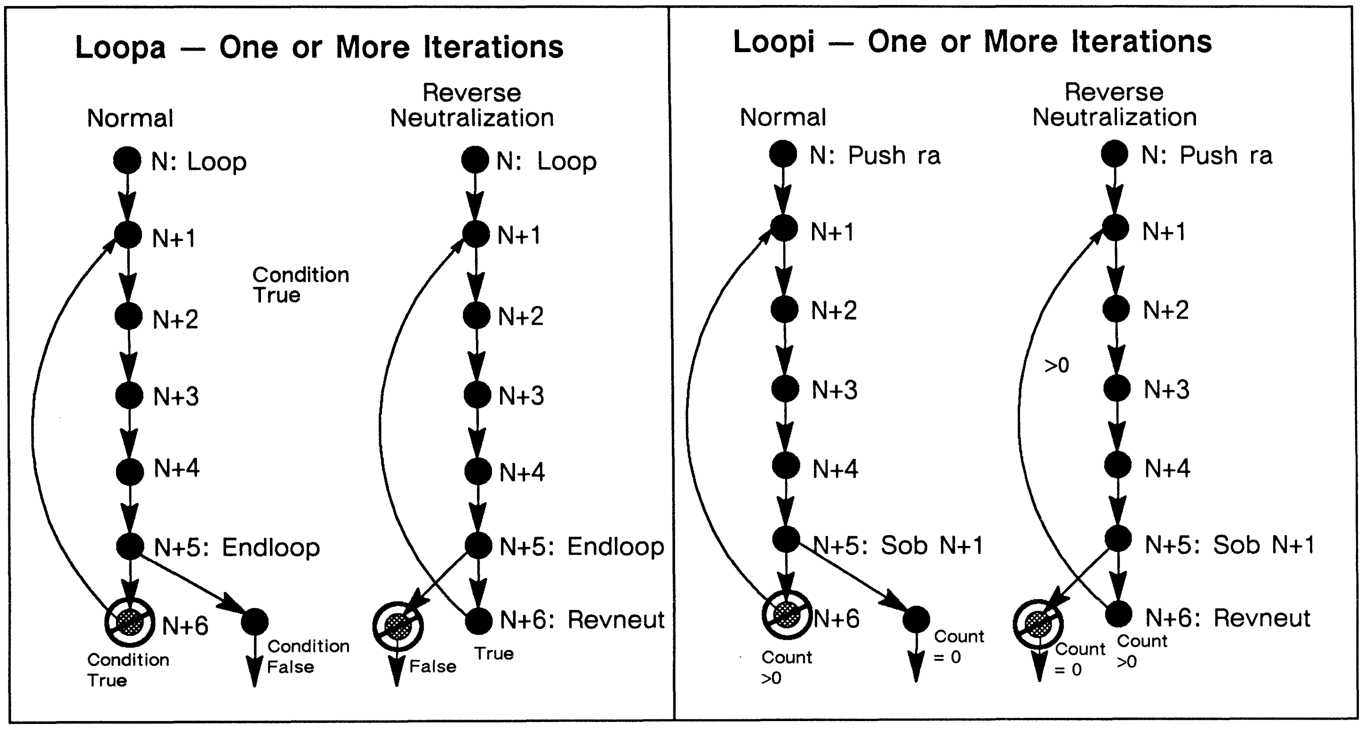


Figure 26. Loopa and Loopi

Loop Control Instructions, continued

LOOP ENTER (PUSH FOLLOWING ADDRESS)

000	00111	RIP/coprocessor
3	5	24

FORMAT

loop

DESCRIPTION

This instruction pushes the address of the currently fetching instruction on the stack. Used to start loopa. Short instruction format.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one.

EXCEPTIONS

Stack overflow

OPERATION

```
if b = 1 then
      NEUT := true;
      b := 0;
else
      NEUT := false;
      tos := tos + 1;
      stack(tos) := cfa;
endif;
cfa := cfa + 1;
AC := cfa;
```

Loop Control Instructions, continued

ABSOLUTE BRANCH TO STACK OR POP ON CONDITION (LOOP AGAIN OR EXIT)

000	01	c	00	RIP/coprocessor
3	2	1	2	24

FORMAT

endloop

DESCRIPTION

If the selected condition is asserted, causes a branch to the contents of the top of stack; otherwise, pops the stack and continues normal sequential execution. If the 24-bit processor/coprocessor field contains a coprocessor instruction, then FPCN is tested, otherwise COND is tested. See the *XL-8237 Data Sheet* for details of the processor/coprocessor instruction field. Used to conditionally end a loopa. (See the brc instruction on page 27 for details of condition code selection.) Short instruction format.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one. If the branch is taken, the b bit of the .ssr is set to one.

EXCEPTIONS

Stack underflow

OPERATION

```
if b = 1 then
    NEUT := true;
    cfa := cfa + 1;
    b := 0;
else
    NEUT := false;
    if {processor operation is coprocessor} then
        cond := FPCN;
    else
        cond := COND;
    endif;
    if c · cond then
        cfa := cfa + 1;
        tos := tos - 1;
        b := 0;
    else
        cfa := stack(tos);
        b := 1;
    endif;
endif;
AC := cfa;
```

Loop Control Instructions, continued

DECREMENT STACK AND BRANCH OR POP IF ZERO (FOR LOOPI END)

000	00001	imm24
3	5	24

FORMAT

sob imm24

DESCRIPTION

The acronym sob stands for subtract-one-and-branch. This instruction subtracts one from the top of stack, replacing result back on the top-of-stack. If the result is non-zero, a branch is caused to the address specified by adding the sign-extended 24-bit immediate imm24 to the address of the currently executing instruction. If the result is zero, the value on the top-of-stack is popped off and discarded, and the normal sequential execution resumes. Useful at bottoms of loops designed to continue a set number of iterations (loopi). Long instruction format.

Note that the initial count must be non-negative, (that is, the high bit must be zero) for the instruction to work properly.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one. If the branch is taken, the b bit of the .ssr is set to one.

EXCEPTIONS

Stack underflow

OPERATION

```
if b = 1 then
    NEUT := true;
    cfa := cfa + 1;
    b := 0;
else
    NEUT := false;
    stack(tos) := stack(tos) - 1;
    if stack(tos) = 0 then
        cfa := cfa + 1;
        tos := tos - 1;
    else
        cfa := cea + (9 dup imm24[23]) || imm24[22..0];
        b := 1;
    endif;
endif;
AC := cfa;
```

Loop Control Instructions, continued

DECREMENT STACK AND BACKWARD BRANCH OR POP IF ZERO (FOR LOOPI END)

001	imm5	RIP/coprocessor
3	5	24

FORMAT

shsob imm5

DESCRIPTION

Same instruction as sob, but performs the branch based on the one-extended field imm5 instead of the 24-bit immediate that sob uses. Used to end a loopi. Short instruction format.

Note that the initial count must be non-negative for the instruction to work properly.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one. If the branch is taken, the b bit of the .ssr is set to one.

EXCEPTIONS

Stack underflow

OPERATION

```
if b = 1 then
    NEUT := true;
    cfa := cfa + 1;
    b := 0;
else
    NEUT := false;
    stack(tos) := stack(tos) - 1;
    if stack(tos) = 0 then
        cfa := cfa + 1;
        tos := tos - 1;
        b := 0;
    else
        cfa := cea + (27 dup 1) || imm5[4..0];
        b := 1;
    endif;
endif;
AC := cfa;
```

Loop Control Instructions, continued

BRANCH AND POP (LOOPI/LOOPA EXIT)

000	00011	imm24
3	5	24

FORMAT

brp imm24

DESCRIPTION

Branches to the address specified by adding the sign-extended 24-bit immediate imm24 to the address of the currently executing instruction. The value on the top of stack is popped off and discarded. Used to unconditionally or prematurely exit a loopa or loopi. Long instruction format.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one. If the branch is taken, the b bit of the .ssr is set to one.

EXCEPTIONS

Stack underflow

OPERATION

```
if b = 1 then
      NEUT := true;
      cfa := cfa + 1;
      b := 0;
else
      NEUT := false;
      cfa := cea + (9 dup imm24[23]) || imm24[22..0];
      tos := tos - 1;
      b := 1;
endif;
AC := cfa;
```

Subroutine Control Instructions Summary

The following instructions are used for subroutine control:

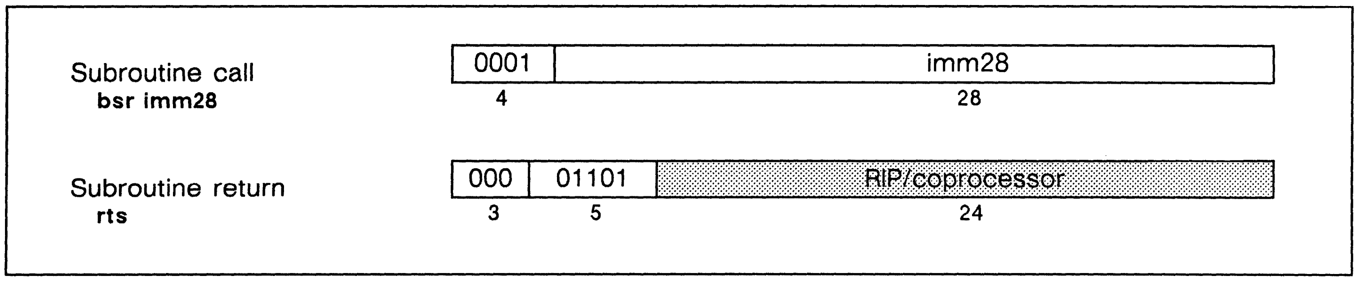


Figure 27. Subroutine control instruction format

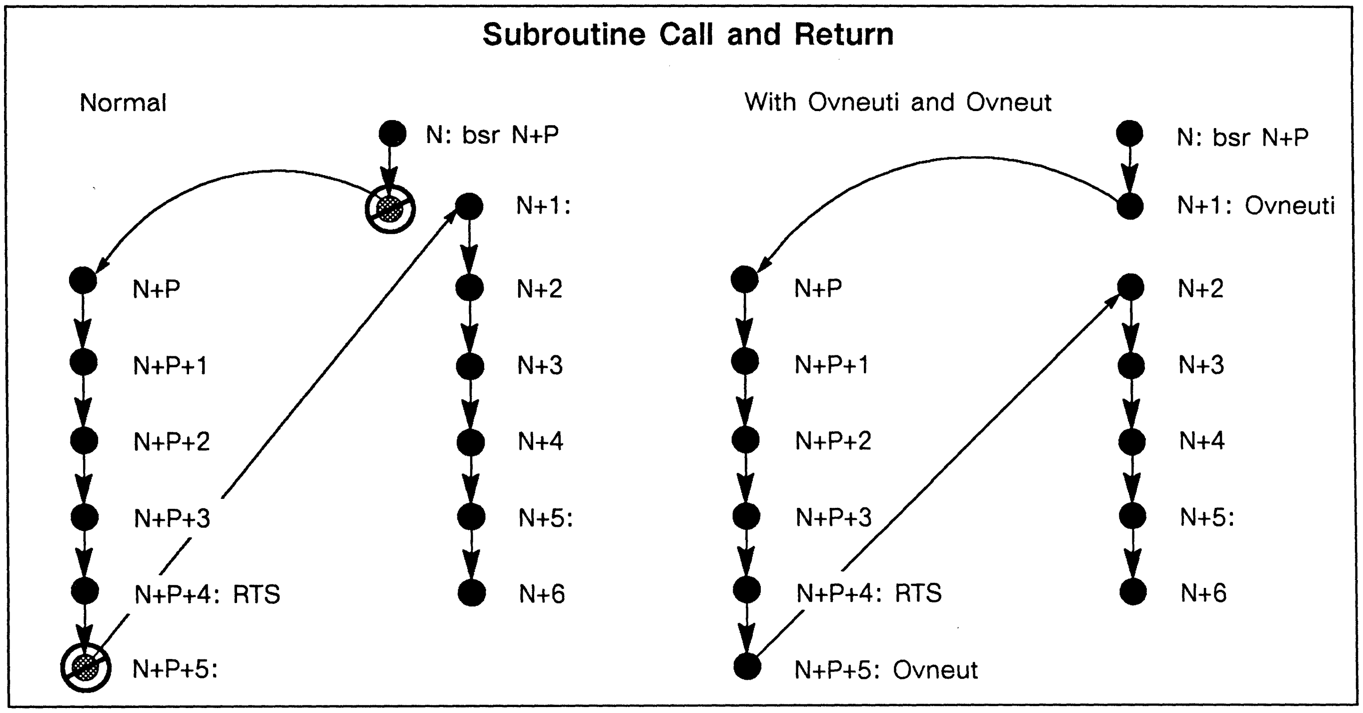


Figure 28. Subroutine call and return

Subroutine Control Instructions, continued

SUBROUTINE CALL

0001	imm28
4	28

FORMAT

bsr imm28

DESCRIPTION

Pushes the address of the currently fetching instruction on the stack and branches to the address specified by the sum of the sign-extended 28-bit immediate imm28 and .cea. Long instruction format.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one. The b bit of the .ssr is set to one.

EXCEPTIONS

Stack overflow

OPERATION

```
if b = 1 then
      NEUT := true;
      cfa := cfa + 1;
      b := 0;
else
      NEUT := false;
      tos := tos + 1;
      stack(tos) := cfa;
      cfa := cea + (5 dup imm28[27]) || imm28[26..0];
      b := 1;
endif;
AC := cfa;
```

Subroutine Control Instructions, continued

SUBROUTINE RETURN

000	01101	RIP/coprocessor
3	5	24

FORMAT

rts

DESCRIPTION

The value on the top of the stack is used as an absolute branch address and discarded. Short instruction format.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one. The b bit of the .ssr is set to one.

EXCEPTIONS

Stack underflow

OPERATION

```
if b = 1 then
      NEUT := true;
      cfa := cfa + 1;
      b := 0;
else
      NEUT := false;
      cfa := stack(tos);
      tos := tos - 1;
      b := 1;
endif;
AC := cfa;
```

Interrupt Control Instructions Summary

The following instructions are used to return from interrupt:

Return from interrupt
rfi0

00000000	x	00101	x
8	8	5	11

Return from interrupt
rfi1

000	00100	RIP/coprocessor
3	5	24

Figure 29. Interrupt control instructions

The following instructions are used to cause software interrupts (traps):

Trap immediate
trapi

00000000	11	x	00010	imm11
8	2	6	5	11

Figure 30. Trap instruction

Interrupt Control Instructions, continued

RETURN FROM INTERRUPT (rfi0)

00000000	x	00101	x
8	8	5	11

FORMAT

rfi0

DESCRIPTION

First of a two-step instruction sequence to restore the RCS state after an interrupt. Restores .cfa from .iea, and enables interrupts by setting the men bit. The rfi0 instruction must be followed by the rfi1 instruction. Long instruction format.

The effect of setting men bit may not take place until one cycle after the rfi0 instruction.

All internal exceptions should be processed before the interrupt handler is exited.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr is a one.

EXCEPTIONS

None

OPERATION

```
if b = 1 then
    cfa := cfa + 1;
    NEUT := TRUE;
else
    NEUT := FALSE;
    if ssr.s = 1 then
        cfa := iea;
        OP := 00100;
        ssr.men := 1
    else
        ssr.prvflg := true;
    endif;
endif;
b := 0;
AC := cfa;
```

Interrupt Control Instructions, continued

RETURN FROM INTERRUPT (rfi1)

000	00100	RIP/coprocessor
3	5	24

FORMAT

rfi1

DESCRIPTION

Second of a two-step instruction sequence to restore the RCS state after an interrupt. Short instruction format. Rfi1 stores the contents of the .ifa to the .cfa and places the contents of the .ifa onto the AC bus. The .ssr bi bit is stored into .ssr b to indicate if the last instruction of the interrupted process was a branch. The rfi1 instruction must always be preceded by the rfi0 instruction. Short instruction format.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one.

EXCEPTIONS

None

OPERATION

```
if b = 1 then
    cfa := cfa + 1;
    b := 0;
    NEUT := FALSE;
else
    NEUT := FALSE;
    if ssr.s = 1 then
        cfa := ifa;
        ssr.b := ssr.bi;
        ssr.s := ssr.si;
        OP := 00111;
    else
        ssr.prvflg := TRUE;
    endif;
endif;
AC := cfa;
```

Trap Instructions

TRAP IMMEDIATE

00000000	11	x	00010	imm11
8	2	6	5	11

FORMAT

trapi imm11

DESCRIPTION

This instruction sets the trpflg bit of the .ssr. If enabled, the RCS causes an interrupt and pushes an 11-bit zero-extended immediate value onto the stack. The trap flag (trpflg) in .ssr is set when a trap occurs.

NEUTRALIZATION

This instruction is neutralized if the b bit of the ssr contains a one.

EXCEPTIONS

Stack overflow and trap

OPERATION

```
if b = 0 then
        ssr.trpflg := 1;
        if (ssr.men = 1) and (ssr.trpen = 1) then
                stack(tos) := 21 dup 0 || imm11[10..0];
                tos := tos + 1;
        endif;
endif;
b := 0;
cfa := cfa + 1;
```

Neutralization Instructions Summary

The RCS has three short-format instructions that change the neutralization of shadow instructions after branching, subroutine calls and returns.

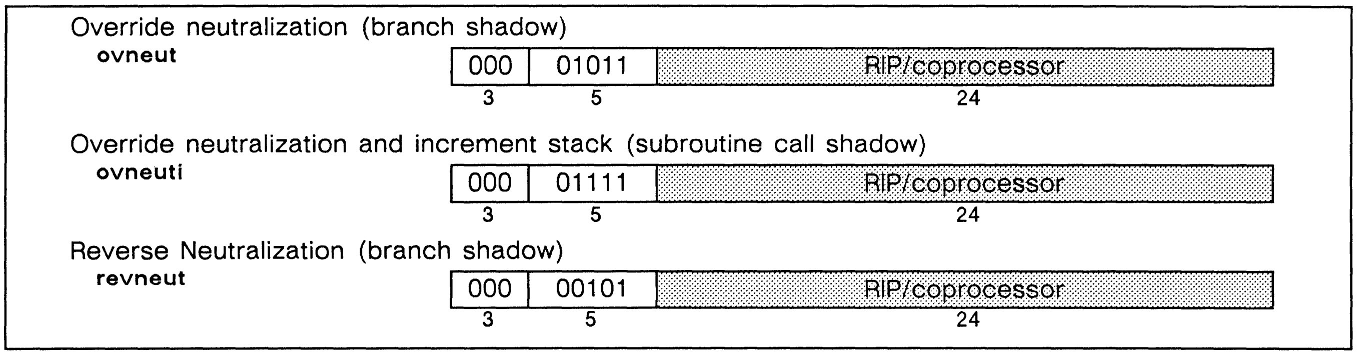

Override neutralization (branch shadow)
ovneut
000
01011
RIP/coprocessor
3
5
24
Override neutralization and increment stack (subroutine call shadow)
ovneuti
000
01111
RIP/coprocessor
3
5
24
Reverse Neutralization (branch shadow)
revneut
000
00101
RIP/coprocessor
3
5
24

Figure 31. Neutralization instruction format

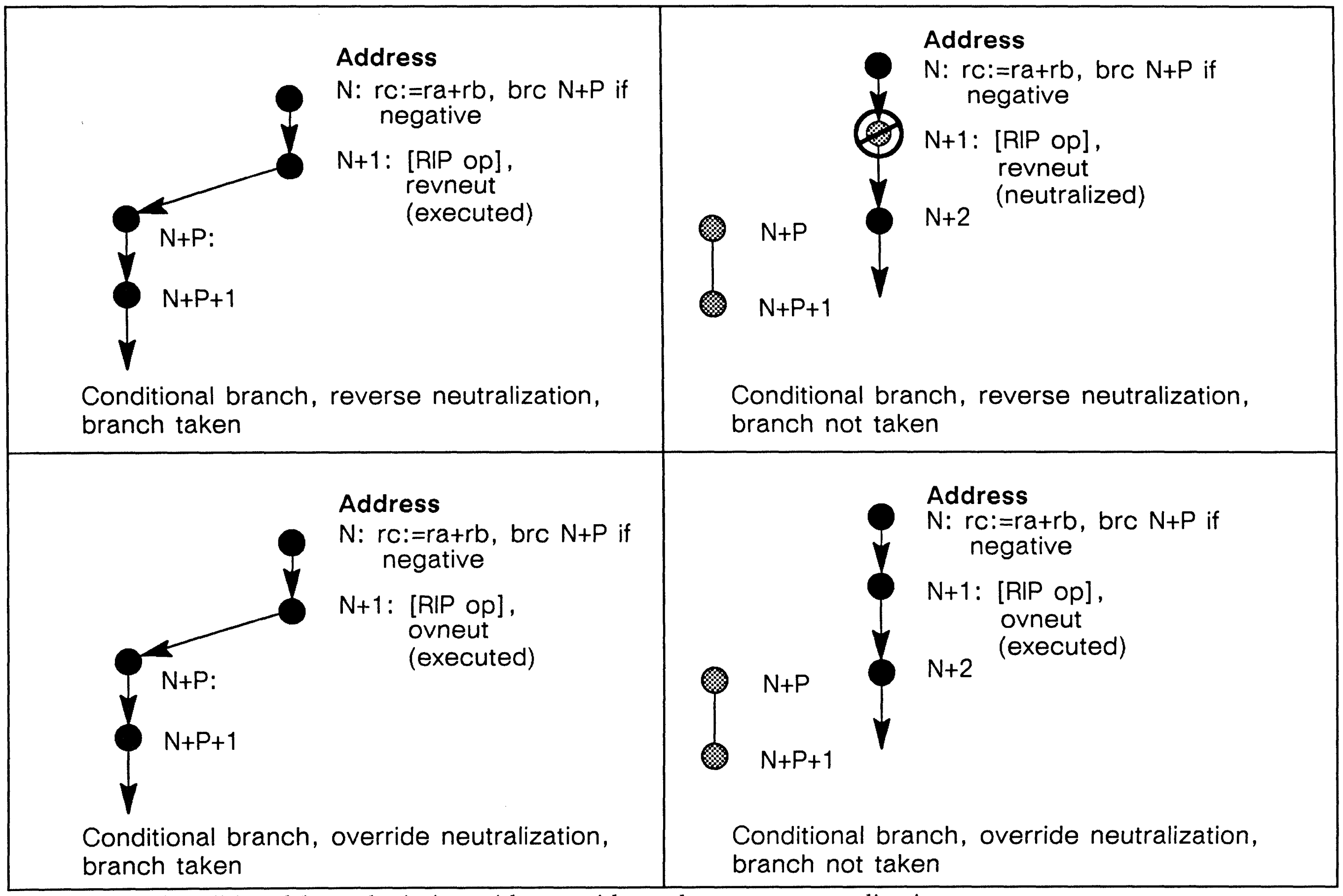

Address
N: rc:=ra+rb, brc N+P if negative
N+1: [RIP op], revneut (executed)
N+P:
N+P+1
Conditional branch, reverse neutralization, branch taken
Address
N: rc:=ra+rb, brc N+P if negative
N+1: [RIP op], revneut (neutralized)
N+2
N+P
N+P+1
Conditional branch, reverse neutralization, branch not taken
Address
N: rc:=ra+rb, brc N+P if negative
N+1: [RIP op], ovneut (executed)
N+P:
N+P+1
Conditional branch, override neutralization, branch taken
Address
N: rc:=ra+rb, brc N+P if negative
N+1: [RIP op], ovneut (executed)
N+2
N+P
N+P+1
Conditional branch, override neutralization, branch not taken

Figure 32. Conditional branch timing with override and reverse neutralization

Neutralization Instructions, continued

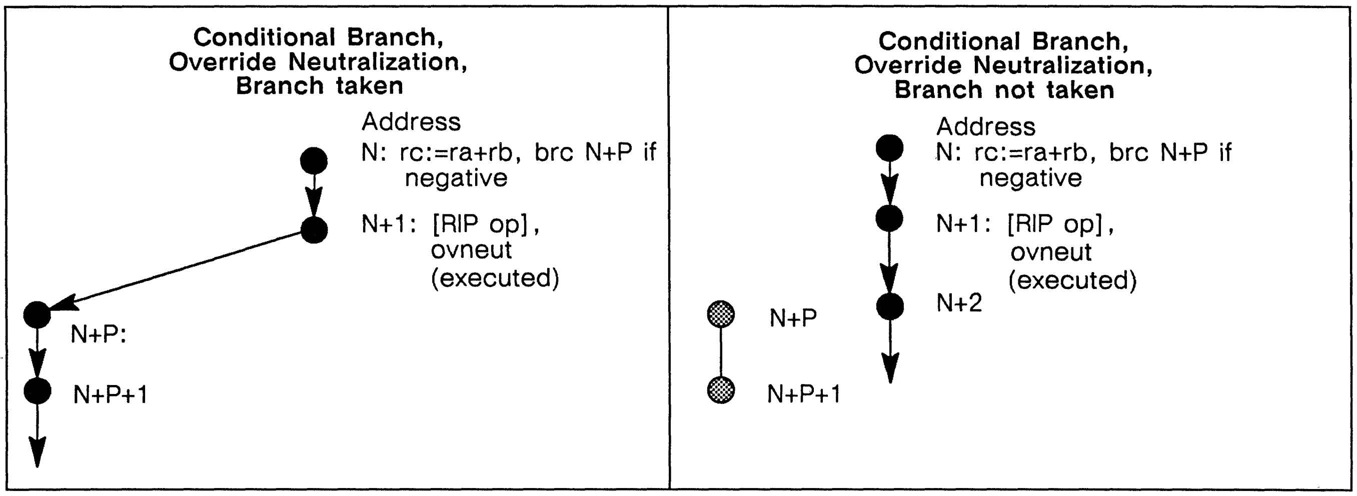

Conditional Branch, Override Neutralization, Branch taken
Address
N: rc:=ra+rb, brc N+P if negative
N+1: [RIP op], ovneut (executed)
N+P:
N+P+1
Conditional Branch, Override Neutralization, Branch not taken
Address
N: rc:=ra+rb, brc N+P if negative
N+1: [RIP op], ovneut (executed)
N+2
N+P
N+P+1

Figure 33. Conditional branch timing with override neutralization

Neutralization Instructions, continued

OVERRIDE NEUTRALIZATION OF BRANCH SHADOW

000	01011	RIP/coprocessor
3	5	24

FORMAT

ovneut

DESCRIPTION

Causes the neutralization effect—which normally cancels the execution of an instruction immediately following a branch (or other transfer-of-control operation)—to be overridden. The instruction is unconditionally executed (unless neutralized due to an interrupt). This instruction has a short format and is placed with the shadow instruction to be executed, immediately following the branch.

NEUTRALIZATION

This instruction is executed regardless of the value of the b field of the .ssr.

EXCEPTIONS

None

OPERATION

```
NEUT := false;
cfa := cfa + 1;
b := 0;
AC := cfa;
```

Neutralization Instructions, continued

OVERRIDE NEUTRALIZATION OF SUBROUTINE CALL SHADOW

000	01111	RIP/coprocessor
3	5	24

FORMAT

ovneuti

DESCRIPTION

This instruction is the same as ovneut, but after executing the shadow instruction, the stack is incremented. Thus a subroutine return is made *not* to the shadow instruction but to the one following it. This instruction has a short format and is placed with the shadow processor operation to be executed, immediately following the subroutine call.

NEUTRALIZATION

This instruction is executed regardless of the b bit of the .ssr.

EXCEPTIONS

None

OPERATION

```
NEUT := false;
stack(tos) := stack(tos) + 1;
cfa := cfa + 1;
b := 0;
AC := cfa;
```

Neutralization Instructions, continued

REVERSE NEUTRALIZATION OF BRANCH SHADOW

000	00101	RIP/coprocessor
3	5	24

FORMAT

revneut

DESCRIPTION

Reverses the neutralization effect that would normally have been applied to an instruction. If it would normally have been neutralized, it is not; if it would normally *not* have been neutralized, it is. Short instruction format.

This instruction is typically used to allow the shadow instruction to be executed as part of a loop, rather than being neutralized. This saves one cycle per iteration.

EXCEPTIONS

None

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a zero.

OPERATION

```
if b = 1 then
      NEUT := false;
else
      NEUT := true;
endif;
cfa := cfa + 1;
b := 0;
AC := cfa;
```

Intrasystem Data Transfer Instructions

TRANSFER WORD TO RCS INTERNAL REGISTER

000	00000	001	ra	ext2	x	from RIP
3	5	3	5	5	11	

000	00000	11	x	ext2	x	from coprocessor
3	5	2	6	5	11	

FORMAT

From RIP

```
mov   ra, .tim          {ext2 = 00110b}
mov   ra, .ssr          {ext2 = 01000b}
mov   ra, .iea          {ext2 = 01010b}
mov   ra, .ifa          {ext2 = 01100b}
mov   ra, .brk          {ext2 = 01110b}
mov   ra, .ibr          {ext2 = 01111b}
```

From Coprocessor

```
mov   .adbus, .tim      {ext2 = 00110b}
mov   .adbus, .ssr      {ext2 = 01000b}
mov   .adbus, .iea      {ext2 = 01010b}
mov   .adbus, .ifa      {ext2 = 01100b}
mov   .adbus, .brk      {ext2 = 01110b}
mov   .adbus, .ibr      {ext2 = 01111b}
```

DESCRIPTION

These instructions cause the RCS to write the contents of the AD bus into the selected internal register. The two instruction formats differ in their effect on the other components in the system. The first set is recognized by the RIP and causes it to drive the AD bus with one register from its register file. The second set allows a coprocessor to drive the AD bus. Long instruction format.

NEUTRALIZATION

These instructions are not neutralized reliably, and therefore must not be put into branch shadows. Furthermore, since interrupts cause an instruction to be neutralized, these instructions must only be used when interrupts are disabled.

Intrasystem Data Transfer Instructions, continued

OPERATION

```
if b = 1 then       { Note: the cancellation of the instruction by NEUT- does not always work. }
    NEUT := TRUE;
else
    NEUT := FALSE;
    if ssr.s = 0 then
        ssr.prvflg := TRUE;
    else
        case ext2 {AD bus is driven by the RIP or a coprocessor} of
            00110b :   tim    := AD;
            01000b :   ssr    := AD;
            01110b :   ibr    := AD;
            01010b :   ;      { no action }
            01100b :   ;      { no action }
            01110b :   ;      { no action }
        endcase;
    endif;
else
    case ext2 of
        00110b :  tim      := AD;
        01000b :  ssr      := AD;
        01110b :  ibr      := AD;
        01010b :  iea      := AD;
        01100b :  ifa      := AD;
        01110b :  brk      := AD;
    endcase;
endif;
b := 0;
cfa := cfa + 1;
AC := cfa;
```

Intrasystem Data Transfer Instructions, continued

TRANSFER WORD FROM RCS INTERNAL REGISTER

000	00000	011	ra	ext2	0	to RIP
3	5	3	5	5	11	

000	00000	11	x	ext2	x	to coprocessor
3	5	2	6	5	11	

FORMAT

To RIP			To Coprocessor		
mov	.tim, ra	{ext2 = 00111b}	mov	.tim, .adbus	{ext2 = 00111b}
mov	.ssr, ra	{ext2 = 01001b}	mov	.ssr, .adbus	{ext2 = 01001b}
mov	.iea, ra	{ext2 = 01011b}	mov	.iea, .adbus	{ext2 = 01011b}
mov	.ifa, ra	{ext2 = 01101b}	mov	.ifa, .adbus	{ext2 = 01101b}

DESCRIPTION

The RCS drives the AD bus with the contents of the designated register. The different formats allow either the RIP or the coprocessor to receive the data. Long instruction format.

NEUTRALIZATION

These instructions are neutralized if the b bit of the .ssr is a one.

EXCEPTIONS

None

Intrasystem Data Transfer Instructions, continued

OPERATION

```
if b = 1 then
      NEUT := TRUE;
else
      NEUT := FALSE;
      if ssr.s = 0 then
            ssr.prvflg := TRUE;
      endif;
      case ext2 of
            00111b :  AD    := tim;
            01001b :  AD    := ssr;
            01011b :  AD    := iea;
            01101b :  AD    := ifa;
      endcase;
endif;
b := 0;
cfa := cfa + 1;
```

Intrasystem Data Transfer Instructions, continued

POP STACK TO RIP REGISTER OR COPROCESSOR

000	00000	010	ra	00000	x	Pop to RIP
3	5	3	5	5	11	

000	00000	101	x	00000	x	Pop to coprocessor
3	5	3	5	5	11	

FORMAT

```
pops ra         {pop to RIP}
or
pops .adbus     {pop to coprocessor}
```

DESCRIPTION

The contents of the top of the stack are driven onto the AD bus. The stack is popped. The RIP recognizes the first format and latches the data from the AD bus. Long instruction format.

NEUTRALIZATION

These instructions are neutralized if the b bit of the .ssr is a one.

EXCEPTIONS

Stack underflow

OPERATION

```
if b = 1 then
    NEUT := TRUE;
else
    NEUT := FALSE;
    AD := stack (tos);          {AD Bus is written to RIP register ra or to coprocessor}
    tos := tos - 1;
endif;
b := 0;
cfa := cfa + 1;
AC := cfa;
```

Intrasystem Data Transfer Instructions, continued

COPY STACK TO RIP REGISTER OR COPROCESSOR

000	00000	010	ra	00001	x	Copy stack to RIP register
3	5	3	5	5	11	

000	00000	101	x	00001	x	Copy stack to Coprocessor
3	5	3	5	5	11	

FORMAT

mov .tos, ra	{copy to RIP}
mov .tos, .adbus	{copy to coprocessor}

DESCRIPTION

The contents of the top of the stack are driven onto the AD bus. The stack is *not* popped. The RIP recognizes the first format and latches the data from the AD bus. Long instruction format.

NEUTRALIZATION

These instructions are neutralized if the b bit of the .ssr is a one.

EXCEPTIONS

None

OPERATION

```
if b = 1 then
      NEUT := TRUE;
else
      NEUT := FALSE;
      AD := stack (tos);
endif;
b := 0;
cfa := cfa + 1;
AC := cfa;
```

Intrasystem Data Transfer Instructions, continued

PUSH RIP REGISTER OR COPROCESSOR ONTO STACK

000	00000	000	ra	00000	x	Push RIP register
3	5	3	5	5	11	

000	00000	100	x	00000	x	Push Coprocessor
3	5	3	5	5	11	

FORMAT

pushs ra	{push RIP register}
pushs .adbus	{push coprocessor}

DESCRIPTION

The contents of the AD Bus are pushed onto the RCS stack. The RIP recognizes the first format and drives the AD bus with the selected register. Long instruction format.

NEUTRALIZATION

These instructions are neutralized if the b bit of the .ssr is a one.

EXCEPTIONS

Stack overflow

OPERATION

```
if b = 1 then
      NEUT := TRUE;
else
      NEUT := FALSE;
      tos := tos + 1;
      stack (tos) := AD;
endif;
b := 0;
cfa := cfa + 1;
AC := cfa;
```

Intrasystem Data Transfer Instructions, continued

TRANSFER WORD TO EXTERNAL REGISTER

000	00000	001	ra	1	extn	x	RIP to external register
3	5	3	5	1	4	11	

000	00000	11	x	1	extn	x	Coprocessor to external register
3	5	2	6	1	4	11	

FORMAT

```
mov  ra, extn
mov  .adbus, extn
```

DESCRIPTION

For all formats the OP bus is driven by the external register number (must be in the range 0000b–1110b). Long instruction format. Other than driving the OP bus, the RCS treats the RIP and coprocessor forms of this instruction as a no-op.

Note: the transfer to and transfer from coprocessor to/from external register instructions have the same format. The actual direction of the transfer is a convention between the coprocessor and the external register and is not specified here.

NEUTRALIZATION

These instructions are neutralized if the b bit of the .ssr is a one.

EXCEPTIONS

Stack underflow

OPERATION

```
if b = 1 then
      NEUT := TRUE;
else
      NEUT := FALSE;
      OP := 1b || extn;
endif;
b := 0;
cfa := cfa + 1;
AC := cfa;
```

Intrasystem Data Transfer Instructions, continued

TRANSFER WORD FROM EXTERNAL REGISTER

000	00000	011	ra	1	extn	x	External register to RIP
3	5	3	5	1	4	11	

000	00000	11	x	1	extn	x	External register to Coprocessor
3	5	2	6	1	4	11	

FORMAT

```
mov   extn, ra
mov   extn, .adbus
```

DESCRIPTION

Takes a word from the AD bus and copies it to a register in the RIP or coprocessor.

For all formats the OP bus is driven by the external register number (must be in the range 0000b–1110b). Other than driving the OP bus, the RCS treats the RIP and coprocessor forms of this instruction as a no-op. Long instruction format.

Note: the transfer to and transfer from coprocessor to/from external register instructions have the same format. The actual direction of the transfer is a convention between the coprocessor and the external register and is not specified here.

NEUTRALIZATION

These instructions are neutralized if the b bit of the .ssr is a one.

EXCEPTIONS

Stack overflow

OPERATION

```
if b = 1 then
      NEUT := TRUE;
else
      NEUT := FALSE;
      OP := 1b || extn;
endif;
b := 0;
cfa := cfa + 1;
AC := cfa;
```

Intrasystem Data Transfer Instructions, continued

RIP/COPROCESSOR LOAD/STORE

101	rd	RIP/coprocessor	Store Coprocessor
110	rd	RIP/coprocessor	Load RIP
111	rd	RIP/coprocessor	Load Coprocessor
000	01001	RIP/coprocessor	Store RIP
3	5	24	

FORMAT

store
load rd
fstore frd (or dstore dfrd, dstorem dfrd, dstorel dfrd)
fload frd (or dload dfrd, dloadm dfrd, dloadl dfrd)

DESCRIPTION

The RIP or coprocessor performs a load or store over the D bus (which is not connected to the RCS). The RCS merely drives the OP bus to indicate RIP load, coprocessor load, RIP store, or coprocessor store. Otherwise, it treats load and store cycles as no-ops. Note that no register is specified for the RIP Store operation; the result of the accompanying RIP operation is simultaneously stored in an RIP register and in the previously addressed word of memory.

NEUTRALIZATION

This instruction is neutralized if the b bit of the .ssr contains a one.

EXCEPTIONS

None

OPERATION

```
if b = 1 then
      NEUT := true;
      b := 0;
else
      NEUT := false;
      if {Store RIP} then
            OP := 01000b or 01001b;*
      endif;
      if {Store Coprocessor} then
            OP := 01100b or 01101b;*
      endif;
      if {Load RIP} then
            OP := 01010b or 01011b;*
      endif;
      if {Load Coprocessor} then
            OP := 01110b or 01111b;*
      endif;
endif;
cfa := cfa + 1;
AC := cfa;
```

* The selection between even or odd OP Bus value is based on the 24-bit RIP operation. If the operation specifies data address generation then the odd value is used, otherwise the even value is used. See *XL-8237 Data Sheet* for address generation instructions.

Instruction Interaction

	cfa	cea	EXECUTING INSTRUCTION		COMMENT
taken branch	N + 1	N	br N + P	⟨ RIP/coprocessor ⟩	branch taken
	N + P	N + 1	⟨ ⟩	⟨ RIP/coprocessor ⟩	neutralized
	N+P+1	N + P	⟨ ⟩	⟨ RIP/coprocessor ⟩	
	N + 1	N	br N + P	⟨ RIP/coprocessor ⟩	branch taken
	N + P	N + 1	revneut	⟨ RIP/coprocessor ⟩	executed
	N+P+1	N + P	⟨ ⟩	⟨ RIP/coprocessor ⟩	
	N + 1	N	br N + P	⟨ RIP/coprocessor ⟩	branch taken
	N + P	N + 1	ovneut	⟨ RIP/coprocessor ⟩	executed
	N+P+1	N + P	⟨ ⟩	⟨ RIP/coprocessor ⟩	
not taken branch	N + 1	N	br N + P	⟨ RIP/coprocessor ⟩	branch not taken
	N + 2	N + 1	⟨ ⟩	⟨ RIP/coprocessor ⟩	executed
	N + 3	N + 2	⟨ ⟩	⟨ RIP/coprocessor ⟩	
	N + 1	N	br N + P	⟨ RIP/coprocessor ⟩	branch not taken
	N + 2	N + 1	revneut	⟨ RIP/coprocessor ⟩	neutralized
	N + 3	N + 2	⟨ ⟩	⟨ RIP/coprocessor ⟩	
	N + 1	N	br N + P	⟨ RIP/coprocessor ⟩	branch not taken
	N + 2	N + 1	ovneut	⟨ RIP/coprocessor ⟩	executed
	N + 3	N + 2	⟨ ⟩	⟨ RIP/coprocessor ⟩	

Branch instructions	Comment
br imm24	unconditional .cea relative
brp	unconditional absolute
shbr imm5	unconditional .cea relative
brc imm5	conditional .cea relative
rts	unconditional absolute
ovneut	branch shadow override
revneut	branch shadow reverse

Figure 34. Effects of branching on code execution

	cfa	cea	EXECUTING INSTRUCTION		COMMENT	STACK
call	N + 1	N	bsr N + F	⟨ RIP/coprocessor ⟩	call	–
	N + P	N + 1	⟨ ⟩	⟨ RIP/coprocessor ⟩	neutralized	N + 1
	N+P+1	N + P	⟨ ⟩	⟨ RIP/coprocessor ⟩	top of subroutine	N + 1
	⋮	⋮	⋮		⋮	
	N+Q+1	N + Q	rts	⟨ RIP/coprocessor ⟩	return – bottom	N + 1
	N + 1	N+Q+1	⟨ ⟩	⟨ ⟩	neutralized	–
	N + 2	N + 1	⟨ ⟩	⟨ RIP/coprocessor ⟩		
	N + 3	N + 2	⟨ ⟩	⟨ RIP/coprocessor ⟩		
call with override	N + 1	N	bsr N + F	⟨ RIP/coprocessor ⟩	call	–
	N + P	N + 1	ovneuti	⟨ RIP/coprocessor ⟩	executed – top of subroutine	N + 1
	N+P+1	N + P	⟨ ⟩	⟨ RIP/coprocessor ⟩		N + 2
	⋮	⋮	⋮		⋮	
	N+Q+1	N + Q	rts	⟨ RIP/coprocessor ⟩	return	N + 2
	N + 2	N+Q+1	ovneut	⟨ RIP/coprocessor ⟩	executed – bottom of subroutine	–
	N + 3	N + 2	⟨ ⟩	⟨ RIP/coprocessor ⟩		–
	N + 4	N + 3	⟨ ⟩	⟨ RIP/coprocessor ⟩		–

Subroutine call instructions	Comment
bsr imm28	unconditional *cea* relative
ovneuti	override neutralization and increment stack

Figure 35. Effects of subroutine calls and returns, with and without call shadow neutralization

Instruction Interaction, continued

	cfa	cea	EXECUTING INSTRUCTION		COMMENT	STACK
	N + 1	N	loop	⟨ RIP/coprocessor ⟩	–	–
	N + 2	N + 1	⟨ ⟩	⟨ RIP/coprocessor ⟩	top of loopa	N + 1
	N + 3	N + 2	⟨ ⟩	⟨ RIP/coprocessor ⟩		N + 1
	⋮	⋮	⋮		⋮	
	N+Q+1	N + Q	endloop	⟨ RIP/coprocessor ⟩	bottom of loopa (not end)	N + 1
loopa	N + 1	N+Q+1	⟨ ⟩	⟨ RIP/coprocessor ⟩	neutralized	N + 1
	N + 2	N + 1	⟨ ⟩	⟨ RIP/coprocessor ⟩	top of loopa	N + 1
	N + 3	N + 2	⟨ ⟩	⟨ RIP/coprocessor ⟩	–	N + 1
	⋮	⋮	⋮		⋮	
	N+Q+1	N + Q	endloop	⟨ RIP/coprocessor ⟩	bottom of loopa (end)	N + 1
	N+Q+2	N+Q+1	⟨ ⟩	⟨ RIP/coprocessor ⟩	execute	–
	N+Q+3	N+Q+2	⟨ ⟩	⟨ RIP/coprocessor ⟩	–	–
	N – P	N–P–1	pushs ra		load stack with count	–
	N–P+1	N – P	⟨ ⟩	⟨ RIP/coprocessor ⟩	top of loop;	2
	N–P+2	N–P+1	⟨ ⟩	⟨ RIP/coprocessor ⟩	–	2
	⋮	⋮	⋮		⋮	
	N	N – 1	⟨ ⟩	⟨ RIP/coprocessor ⟩	–	2
	N + 1	N	sob		bottom of loopi	2
loopi	N – P	N + 1	⟨ ⟩	⟨ RIP/coprocessor ⟩	neutralize	1
	N–P+1	N – P	⟨ ⟩	⟨ RIP/coprocessor ⟩	–	1
	N–P+2	N–P+1	⟨ ⟩	⟨ RIP/coprocessor ⟩	–	1
	⋮	⋮	⋮		⋮	
	N	N – 1	⟨ ⟩	⟨ RIP/coprocessor ⟩	–	1
	N + 1	N	sob		bottom of loopi – end	1
	N + 2	N + 1	⟨ ⟩	⟨ RIP/coprocessor ⟩	execute	0
	N + 3	N + 2	⟨ ⟩	⟨ RIP/coprocessor ⟩	–	0
	N + 4	N + 3	⟨ ⟩	⟨ RIP/coprocessor ⟩	–	

Loop instructions	Comment
loop	loopa top
endloop	loopa bottom
shsob	loopi bottom
sob	loopi bottom
brp	loopa/loopi exit

Figure 36. Effects of looping instructions. Loopa uses the stack for the loop address and loopi uses the stack for the loop count.

Instruction Interaction, continued

	cfa	cea	EXECUTING INSTRUCTION	COMMENT	IFA	IEA	MASTER INTERRUPT ENABLE
	N - 1	N - 2	⟨seq⟩, ⟨ RIP/coprocessor ⟩	application	–	–	1
	N	N - 1	⟨seq⟩, ⟨ RIP/coprocessor ⟩		–	–	1
	N + 1	N	⟨seq⟩, ⟨ RIP/coprocessor ⟩	interrupt	–	–	1
	I	N + 1	⟨seq⟩, ⟨ RIP/coprocessor ⟩	neutralized	N + 2	N + 1	0
	I + 1	I	⟨seq⟩, ⟨ RIP/coprocessor ⟩	interrupt routine	N + 2	N + 1	0
interrupt	I + 2	I + 1	⟨seq⟩, ⟨ RIP/coprocessor ⟩		N + 2	N + 1	0
	⋮	⋮		⋮	⋮	⋮	⋮
	J	J - 1	rfi0		N + 2	N + 1	0
	N + 1	J	rfi1 , asrtadr	restart load/store	N + 2	N + 1	0 or 1
	N + 2	N + 1	⟨seq⟩, ⟨ RIP/coprocessor ⟩	application	–	–	1
	N + 3	N + 2	⟨seq⟩, ⟨ RIP/coprocessor ⟩	application	–	–	1

Figure 37. Interrupt sequence, showing the transition from the application program to the interrupt handler and back.

OP Output Bus Operation

The OP bus is provided to indicate to external logic the current operation of the system. Its primary use is to indicate the status of the D and AD buses. On each cycle, the RCS examines the state of the system and the current instruction (the entire 32-bit instruction is examined even if the current RCS instruction has short format) and selects the appropriate OP bus code. Figure 38 lists OP bus codes.

DEFAULT

The default code indicates that nothing interesting is happening during this cycle. This code is asserted whenever none of the others is asserted. This code is also asserted during any cycle where STALL– was asserted at the end of the previous cycle

DATA ADDRESS

These codes indicate that the RIP is currently executing an address generation instruction, that is, any cycle in which the AD bus will be driven with a data address.

LOAD RIP DATA

The RIP is executing a load data instruction. The D bus should be driven by the memory subsystem with the correct data. This code can also occur if the RIP executes an align (byte align for load data) instruction, even though this instruction loads no data. Memory-mapped peripherals can be fooled by this false indication. For this reason we recommend that external register I/O be used instead of memory-mapped I/O.

LOAD COPROCESSOR DATA

The coprocessor is executing a load data instruction. The D bus should be driven by the memory subsystem with the correct data. Most systems treat these codes the same as the load processor data codes.

OP+	DESCRIPTION OF CURRENT INSTRUCTION
00000	default
00001	data address*
00010	interrupt acknowledge, or reset
00011	reserved
00100	return from interrupt 0
00101	reserved
00110	reserved
00111	return from interrupt 1*
01000	store RIP data
01001	store RIP data and data address*
01010	load RIP data†
01011	load RIP data and data address* †
01100	store coprocessor
01101	store coprocessor and data address*
01110	load coprocessor
01111	load coprocessor and data address*
10000	select external register #0
⋮	⋮
11101	select external register #13
11110	select external register #14
11111	reserved

* The external data address register should be clocked on these combinations only.
† These codes can also be generated by the byte align for load instruction (align) in the RIP, even though this instruction loads no data. For this reason, memory-mapped peripherals should be used with caution or not at all.

Figure 38. OP bus decoding

OP Output Bus Operation, continued

STORE RIP DATA

The RIP is executing a store data instruction. For details of the timing of the D bus and WREN– bus outputs relative to the store data instruction, refer to the *XL-8237 Data Sheet.*

STORE COPROCESSOR DATA

The coprocessor is executing a store coprocessor data instruction. For timing details refer to the *XL-8237* and *XL-3132 Data Sheets.*

INTERRUPT ACKNOWLEDGE

The RCS is currently honoring an interrupt request. This code appears during the cycle in which the AC bus address reflects the first instruction of the interrupt routine. The instruction that would normally have been executed during this cycle is always neutralized.

RETURN FROM INTERRUPT 0

The current instruction is rfi0.

RETURN FROM INTERRUPT 1

The current instruction is rfi1. If the RIP is executing an address generation instruction, then this instruction will also drive out the internal .adr register onto the AD Bus to allow the external address register to be updated to reflect its contents before the interrupt is serviced. Thus, this code is also used to clock the external .adr register. To accomplish this, the first eight most significant bits of the instruction specify the RCS rfi1 instruction, and the other 24 bits specify an RIP asrtadr instruction. See the *XL-8237 Data Sheet* for more details.

SELECT EXTERNAL REGISTER

The current instruction is a move to/from external register instruction. The AD bus in this cycle will be used for this intra-processor transfer.

Development Tools

WEITEK provides a family of software tools to aid applications development and debugging, using the XL-8236 and its companion processors, the XL-8237 32-bit raster image processor and the XL-3232 32-bit graphics floating point data path unit.

The XL-8236 is part of WEITEK's XL-Series of processor, and is largely compatible with them. All devices in WEITEK's XL-Series use the same development tools.

HIGH-LEVEL LANGUAGE COMPILERS

The XL-Series supports industry-standard implementations of C and FORTRAN 77 compilers. Industry-standard implementations allow existing programs to be ported to the XL-Series without modification. These compilers all share an optimizing code generator which employs optimization techniques found on mainframe compilers, as well as a parallelizing instruction scheduler that allows the XL's execution units to run in parallel.

For some algorithms (such as key graphics operations) code efficiency can be increased by returning to assembly code. These hand-coded routines can be linked with software modules written in a high-level language.

POSTSCRIPT-COMPATIBLE INTERPRETER

WEITEK supplies its HyperScript interpreter, a PostScript-compatible interpreter that offers form, fit, function, and image compatibility with that offered by Adobe Systems, Inc. Users of the XL-8200 chip family gain a royalty-free right to object code.

WEITEK also supports third-party page description languages through development tools and optimized floating-point and graphics libraries.

The interpreter supports both Bitstream FontWare and URW's NIMBUS font-scaling software. Fonts are fully compatible with Adobe Font Metrics and are represented in Bezier outline form.

COMPLETE DEVELOPMENT SYSTEM SUPPORT

The design of an XL-Series-based product is simplified by the XL software and hardware development tools. The application programmer is able to develop and debug software on a VAX or PC/AT system with the XL-Series Software Development Environment, which includes a software simulator. For the hardware designer, complete engineering documentation is available.

The design of raster image processors is also facilitated by a graphics development system which is composed of a RIP board with the XL-8200, 3 Mbytes of page buffer and font memory, 256 kwords of code memory for the interpreter, PC/AT system interface, and Canon LBP-SX video interface card. This graphics development system provides a stable hardware environment on which PDLs can be debugged independently of the final target hardware.

Design Requirements

Several special steps must be taken to guarantee that your XL-8236 design will function correctly with present and future silicon. These steps must be taken if your design is to work correctly:

A latch, not a register, should be used to latch the code address (AC bus). Future XL-Series silicon may include an on-chip latch.

Coprocessor instructions are reserved for future expansion.

Memory-mapping should not be used: external register I/O should be used instead.

The special treatment of rfi0 and rfi1 must be implemented (see page 11).

The transfer data to RCS internal register instructions must be kept out of branch shadows and used only when interrupts are disabled.

Set all fields marked "x" to zero in instructions which contain them. This assures that operations which are added in future designs will not modify the function of current instructions.

Pins marked "NC" (not connected) on the pin configuration diagram may be defined as *signal pins* in future enhancements to the RIP. Therefore, to preserve future upward compatibility, these pins should indeed be left unconnected.

Pins marked "TIE HIGH" or "TIE LOW" should be tied to VCC or GND. These pins may be redefined in the future as signal pins, in which case you may no longer want them tied high or low. Thus we recommend that they be tied through traces rather than directly to VCC or ground planes.

Absolute Maximum Ratings

Supply voltage	−0.5 to 7.0 V
Input voltage	−0.5 to V_{CC}
Output voltage	−0.5 to 5.5 V_{CC}
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

Figure 39. Absolute maximum ratings

Recommended Operating Conditions

PARAMETER		COMMERCIAL			UNIT
		MIN	NOM	MAX	
V_{CC}	Supply voltage	4.75	5.0	5.25	V
I_{OH}	High-level output current			-1.0	mA
I_{OL}	Low-level output current			4.0	mA
T_{CASE}	Operating case temperature	0		85	°C

Figure 40. Recommended operating conditions

DC Specifications

PARAMETER		TEST CONDITIONS	COMMERCIAL		UNIT
			MIN	MAX	
V_{IH}	High-level input voltage	V_{CC} = MIN	2.0		V
V_{IHC}	High-level input voltage for CLK only	V_{CC} = MIN	2.4		V
V_{IL}	Low-level input voltage	V_{CC} = MAX		0.8	V
V_{ILC}	Low-level input voltage for CLK only	V_{CC} = MAX		0.8	V
V_{OH}	High-level output voltage	V_{CC} = MIN, I_{OH} = −1.0 mA	2.8		V
V_{OL}	Low-level output voltage	V_{CC} = MIN, I_{OL} = 4.0 mA		0.4	V
I_{LI}	Input leakage current	V_{CC} = MAX, V_{IN} = 0 − V_{CC}		±10	µA
I_{LO}	Output leakage current (output disabled)	V_{CC} = MAX, V_{OUT} = 0 − V_{CC}		±10	µA
I_{CC}	Standby current	V_{CC} = MAX, DC Conditions TTL inputs		150	mA
I_{CC}	Switching current	V_{CC} = MAX, T_{CY} = MIN TTL inputs		250	mA
C_{IN}	Input capacitance†	T_A = 25° C		8	pF
C_{CLK}	Clock capacitance†	f = 1 MHz		20	pF
C_{OUT}	Output capacitance†	V_{CC} = 5.0 V		10	pF

† Capacitance not tested

Figure 41. DC specifications

AC Specifications

AC TEST CONDITIONS:				
V_{CC} = MIN	V_{IH} = 3.5V 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	XL-8236-40		XL-8236-20		XL-8236-10		SEE FIGURE	COMMENTS
	MIN	MAX	MIN	MAX	MIN	MAX		
T_{CY} CLK cycle time	120		200		350		44, 48	
T_{CH} CLK HIGH time	55		90		165		44, 48	
T_{CL} CLK LOW time	55		90		165		44, 48	
T_R CLK rise time		5		5		5	48	
T_F CLK fall time		5		5		5	48	
T_{S1} Set-up time for control inputs COND+, STALL−, EXT1−, EXT2−, EXT4−, RESET−, FPCN+	13		20		35		43	
T_{S2} Set-up time for code bus and AD[31..3]	25		30		35		43	
T_{S3} Set-up time for AD[2..0] during RIP address generation only	35		40		45		43	
T_{H1} Input hold time	3		3		3		43	For all inputs except C bus
T_{H2} Input hold time	5		5		5		43	C bus inputs only
T_1 COND input to AC valid		40		60		80	43	COND and STALL− are examined during T_{CL} and are thus not specified with respect to the rising edge of CLK
T_2 STALL− input to AC valid		40		60		80	43	
T_4 EXT1−, EXT2−, EXT4−, RESET−, FPCN+ input to AC valid		45		65		85	43	
T_5 CLK falling edge to AC Bus output		65		100		150	43	
T_6 CLK to AD+ turn-on	15		15		15		43	
T_7 AD+ bus turn-off time		55		60		65	43	
T_8 CLK to AD+ bus valid		95		160		300	43	
T_9 CLK to OP+ valid		35		55		70	43	If certain instruction combinations are avoided.
T_{9A} CLK to OP+ valid		55		65		85	43	General case. See p. 67
T_{11} CLK to NEUT− valid		40		60		80	43	
T_{VO} Output valid time	5		5		5		43	
T_{ZO} Output enable time		30		40		50	45	
T_{OZ} Output disable time		30		40		50	45	
All units in nanoseconds								

Figure 42. AC specifications: guaranteed switching characteristics over commercial temperature range and operating conditions. Contact your WEITEK sales representative for XL-8236-60 specifications

Timing Description

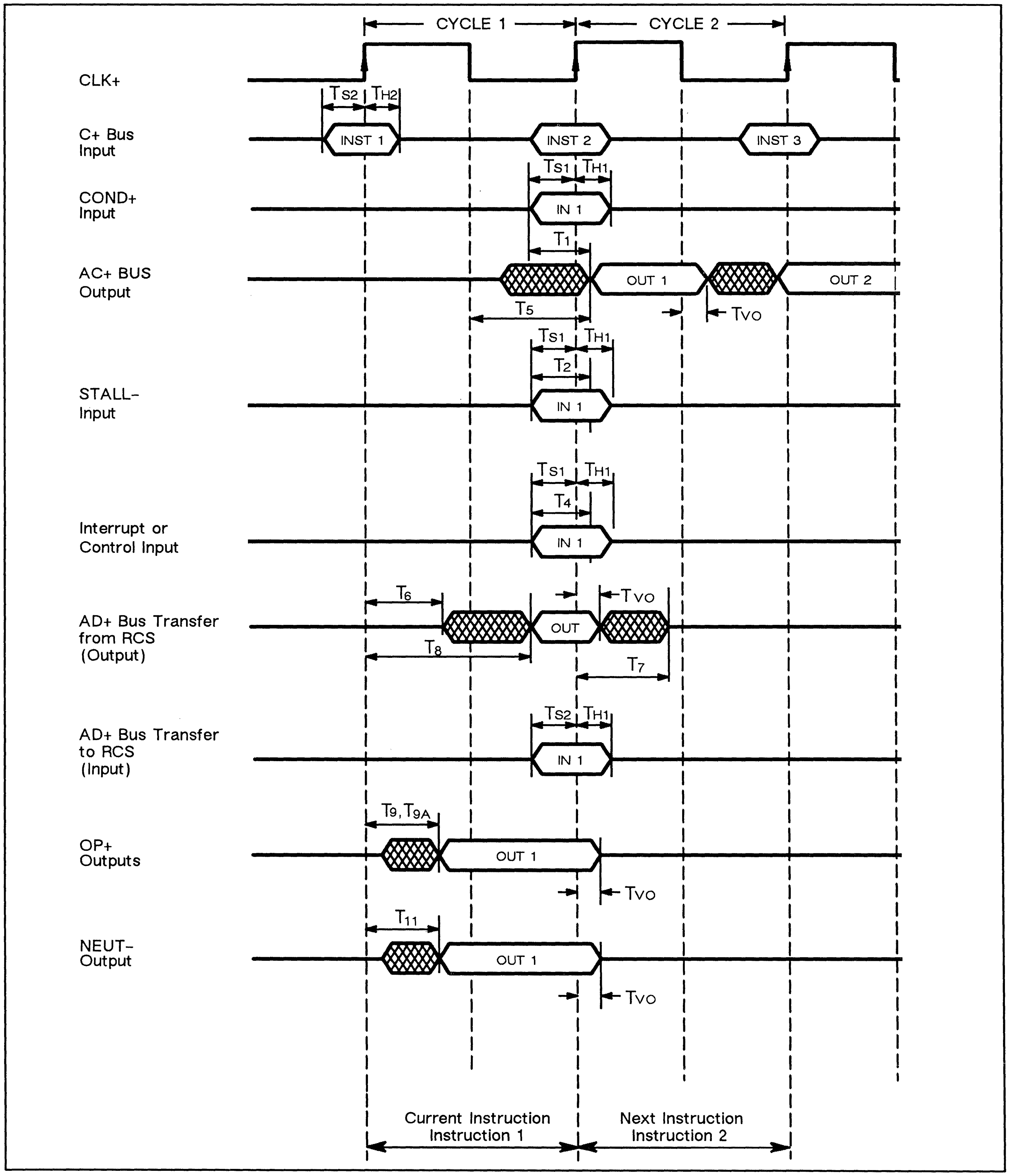


Figure 43. Timing diagram

Timing Description, continued

Signal timing is shown in figure 43. At the beginning of clock cycle 1, an instruction is received from the edge-triggered code register connected to the C bus and executed. If the instruction is a transfer from the RCS, the AD bus is driven by an internal register at time T_8 after the start of the cycle. If the instruction is a transfer to the RCS, the data is received by an edge-triggered register on the AD bus at the end of the cycle. The AD output drivers have an turn-on time from the CLK edge of T_6.

If the previous instruction is a taken branch, the b bit of the .ssr will be set. This can activate the NEUT– output line on the current instruction cycle at time T_{11}, after the beginning of the cycle.

The OP outputs are driven at T_9 or T_{9A}. The timing for T_9 assumes that certain instruction combinations will not be used in the code; specifically, that load, store, rfi0, input, and output instructions will never be placed either in the shadow of a flow-of-control instruction or at the target of a flow-of-control instruction. The T_{9A} specification assumes that no such care is taken. Transfer-of-control instructions include br, shbr, brstkp, endloop, sob, shsob, brp, bsr, and rts; any instruction that causes the program counter to do anything besides increment is a transfer-of-control instruction. The clocking of the external AD bus register on the RIP must work with the general (slower) case.

The XL-Series compilers never generate code that violates this rule. A program to check for violations of this rule in user-written assembly-code routines is included with the XL-8200 development system.

The code address which appears on the AC bus depends upon a number of control inputs and settles at times T_1, T_2, T_4, or T_5 after the corresponding control input is changed. These control inputs are received by level-sensitive latches that are transparent during the second part of the cycle. These control inputs include EXT1–, EXT2-, EXT4–, RESET–, STALL–, COND+, and FPCN+.

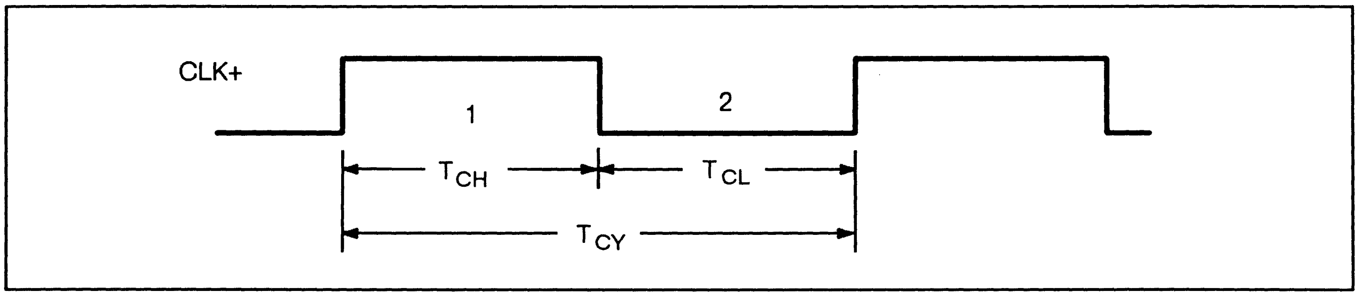
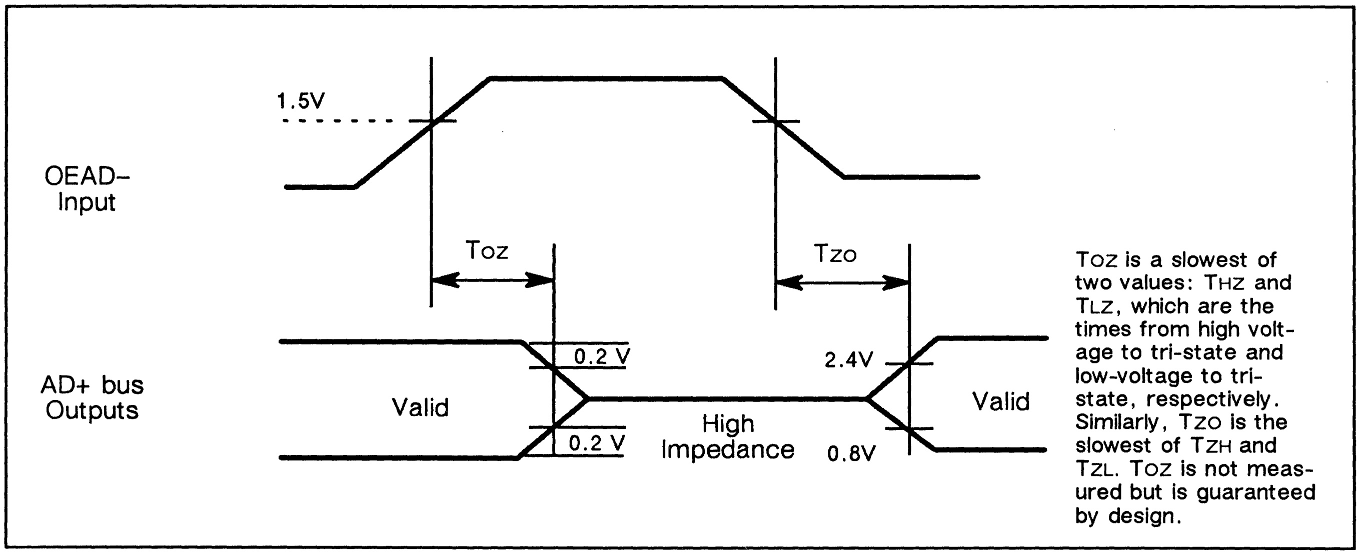


Figure 44. Clock timing



Toz is a slowest of two values: THZ and TLZ, which are the times from high voltage to tri-state and low-voltage to tri-state, respectively. Similarly, TZO is the slowest of TZH and TZL. TOZ is not measured but is guaranteed by design.

Figure 45. Tri-state timing

I/O Characteristics

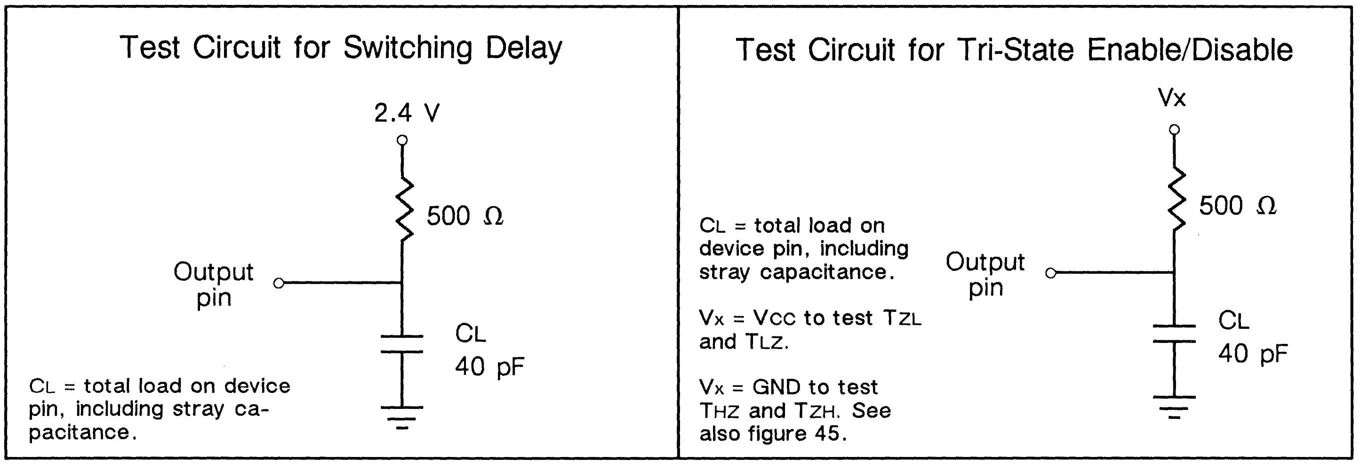

Test Circuit for Switching Delay
2.4 V
500 Ω
Output pin
CL
40 pF
CL = total load on device pin, including stray capacitance.
Test Circuit for Tri-State Enable/Disable
Vx
500 Ω
CL = total load on device pin, including stray capacitance.
Vx = Vcc to test TZL and TLZ.
Vx = GND to test THZ and TZH. See also figure 45.
Output pin
CL
40 pF

Figure 46. Test Load for Delay Measurement

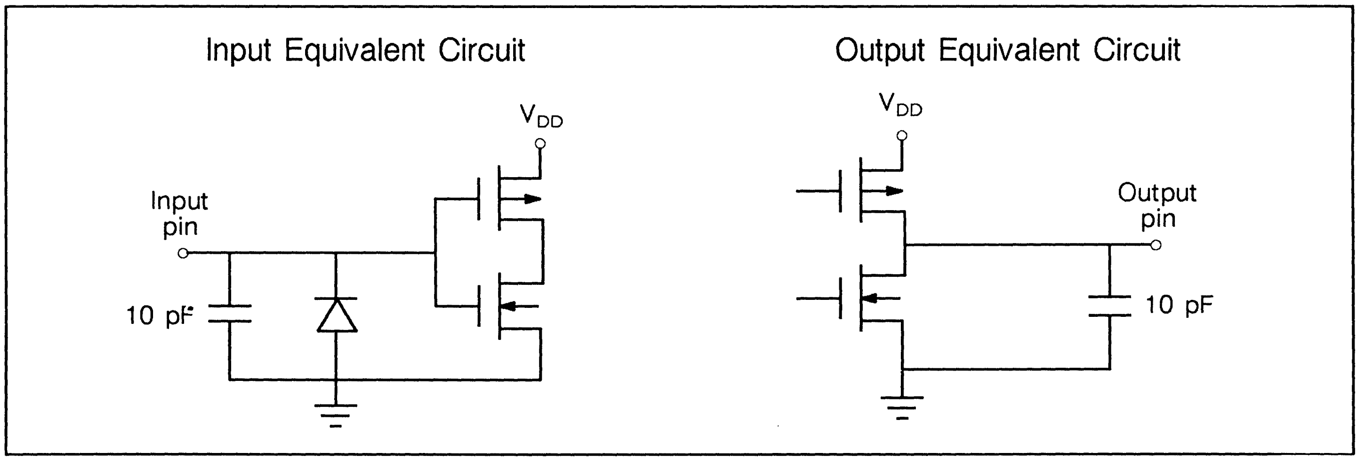


Figure 47. Input and Output Equivalent Circuits

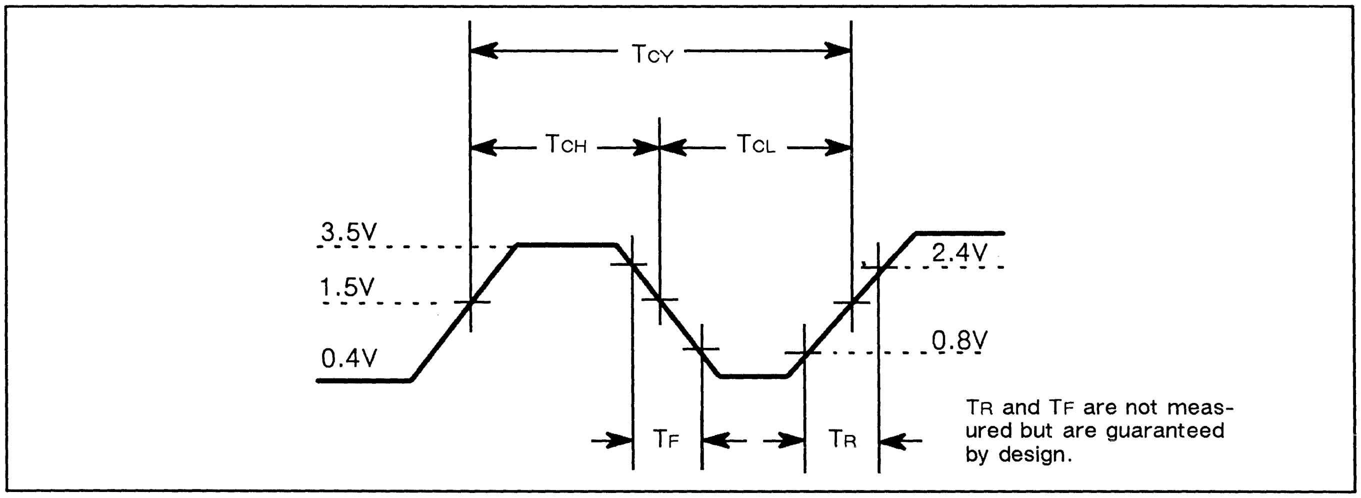
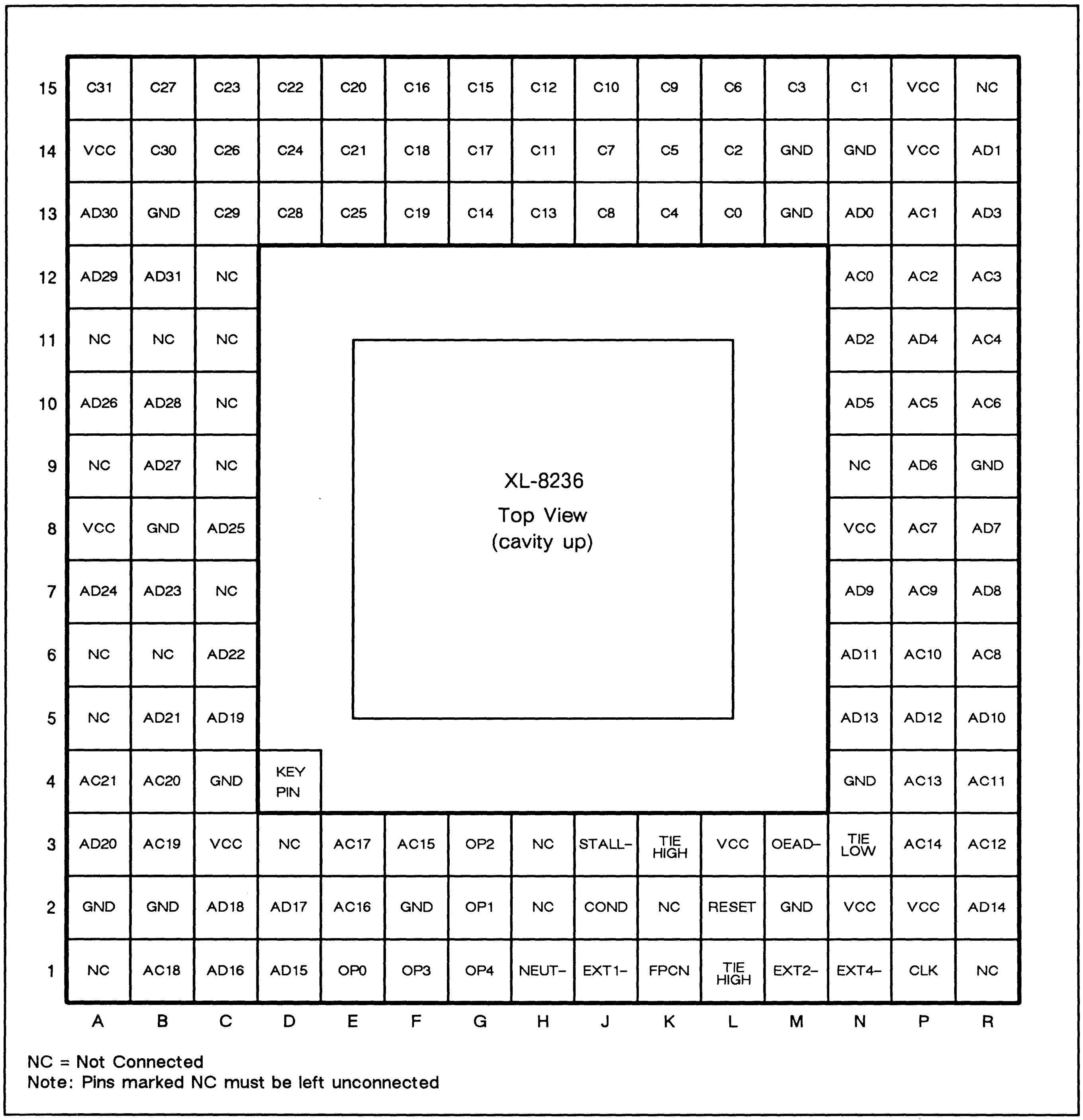


Figure 48. Clock timing

Pin Configuration

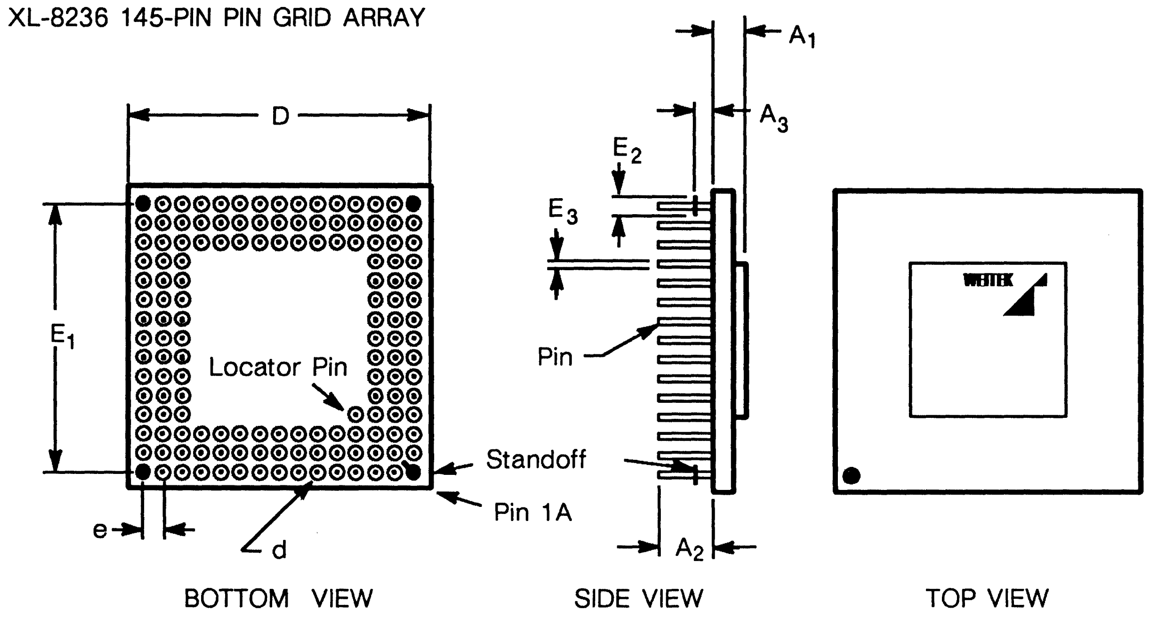


	A	B	C	D	E	F	G	H	J	K	L	M	N	P	R
15	C31	C27	C23	C22	C20	C16	C15	C12	C10	C9	C6	C3	C1	VCC	NC
14	VCC	C30	C26	C24	C21	C18	C17	C11	C7	C5	C2	GND	GND	VCC	AD1
13	AD30	GND	C29	C28	C25	C19	C14	C13	C8	C4	C0	GND	AD0	AC1	AD3
12	AD29	AD31	NC										AC0	AC2	AC3
11	NC	NC	NC										AD2	AD4	AC4
10	AD26	AD28	NC										AD5	AC5	AC6
9	NC	AD27	NC										NC	AD6	GND
8	VCC	GND	AD25					XL-8236 Top View (cavity up)					VCC	AC7	AD7
7	AD24	AD23	NC										AD9	AC9	AD8
6	NC	NC	AD22										AD11	AC10	AC8
5	NC	AD21	AD19										AD13	AD12	AD10
4	AC21	AC20	GND	KEY PIN									GND	AC13	AC11
3	AD20	AC19	VCC	NC	AC17	AC15	OP2	NC	STALL–	TIE HIGH	VCC	OEAD–	TIE LOW	AC14	AC12
2	GND	GND	AD18	AD17	AC16	GND	OP1	NC	COND	NC	RESET	GND	VCC	VCC	AD14
1	NC	AC18	AD16	AD15	OP0	OP3	OP4	NEUT–	EXT1–	FPCN	TIE HIGH	EXT2–	EXT4–	CLK	NC

NC = Not Connected
Note: Pins marked NC must be left unconnected

Figure 49. Pin configuration

Physical Dimensions



Symbol	INCHES		MM	
	MAX	MIN	MAX	MIN
A1	0.135	0.080	3.43	2.03
A2	0.210	0.175	5.33	4.46
A3	0.080	0.040	2.03	1.14
D	1.657	1.555	42.1	39.4
E1	0.140 TYP		3.56 TYP	
E2	0.050 TYP		1.27 TYP	
E3	0.020	0.016	0.51	0.41
d	0.075	0.035	1.91	0.89
e	0.100 TYP		2.54	

Figure 50. XL-8236 physical dimensions

Ordering Information

PACKAGE TYPE	SPEED GRADE	TEMP. RANGE (CASE)	ORDER NUMBER
145-pin plastic PGA	–10	T = 0 to +85°C	XL-8236-010-GPU
145-pin plastic PGA	–20	T = 0 to +85°C	XL-8236-020-GPU
145-pin plastic PGA	–40	T = 0 to +85°C	XL-8236-040-GPU
145-pin plastic PGA	–60	T = 0 to +85°C	XL-8236-060-GPU

PACKAGE TYPE	SPEED GRADE	TEMP. RANGE (CASE)	ORDER NUMBER
145-pin ceramic PGA	–10	T = 0 to +85°C	XL-8236-010-GCU
145-pin ceramic PGA	–20	T = 0 to +85°C	XL-8236-020-GCU
145-pin ceramic PGA	–40	T = 0 to +85°C	XL-8236-040-GCU
145-pin ceramic PGA	–60	T = 0 to +85°C	XL-8236-060-GCU

Figure 51. Ordering information

Revision Summary

T7 was corrected in figure 43

Input hold times increased from 3 ns to 5 ns

The –60 part grade is now available

The package designator for plastic parts in the order number changed from "PGCU" to "GPU"

Figure 52. Revision summary

Index

Index, continued

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

Name ______________________ Title ______________________

Company ______________________ Phone ______________________

Address ______________________

Comments ______________________

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

APPLICATION:

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

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

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

WEITEK use: Rec'd Out TPT Source: DS

Status

WEITEK XL-8236 22-BIT RASTER CODE SEQUENCER
Please Comment On The Quality Of This Data Sheet.

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

Fold, Staple and Mail to Weitek Corp.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1374 SUNNYVALE, CA

POSTAGE WILL BE PAID BY ADDRESSEE

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

ATTN: Ed Masuda

WEITEK

WEITEK'S CUSTOMER COMMITMENT:

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

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

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

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

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

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

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

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

[illegible]

Art Collmeyer
President

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