XL-SERIES OVERVIEW

PRELIMINARY DATA
May 1987

Features

HIGH-SPEED CMOS PROCESSORS

XL-8000: 8 MIPS integer processor

XL-8032: 8 MIPS, 5 MFLOPS single-precision floating point processor

XL-8064: 7 MIPS, 6 MFLOPS double-precision floating point processor

RISC ARCHITECTURE

Single-cycle execution

Three-address, register-to-register instructions

32-word register files

Separate code and data memories for high memory bandwidth

Vectored interrupts

DEVELOPMENT TOOLS

Industry-standard C and FORTRAN 77 compilers

Assembler, linker, and debugger

Functional and architectural simulators

Development system

RICH INSTRUCTION SET

Full set of arithmetic and logical functions

Single-cycle bitwise merge, field extract, field insert, and field merge

Pre- and post-incrementing indexed addressing

Sophisticated program control instructions

Description

The XL Series is a family of three VLSI processors: the XL-8000, a high-speed 32-bit processor; the XL-8032, a single-precision floating point processor with all the features of the XL-8000; and the XL-8064, a double-precision floating point processor with all the features of the XL-8000 plus a full implementation of 32- and 64-bit IEEE arithmetic.

All XL-Series processors have a 32-element register file, a 33-element program control stack, an on-chip integer multiply/divide unit, and a complete set of arithmetic, bit manipulation, address generation, and control instructions. The XL-8032 and XL-8064 also have floating point units with their own on-chip register files.

These processors give the performance of bit-slice components, and have a full complement of development tools, including C and FORTRAN 77 compilers, an assembler, a development system, and hardware and software simulators.

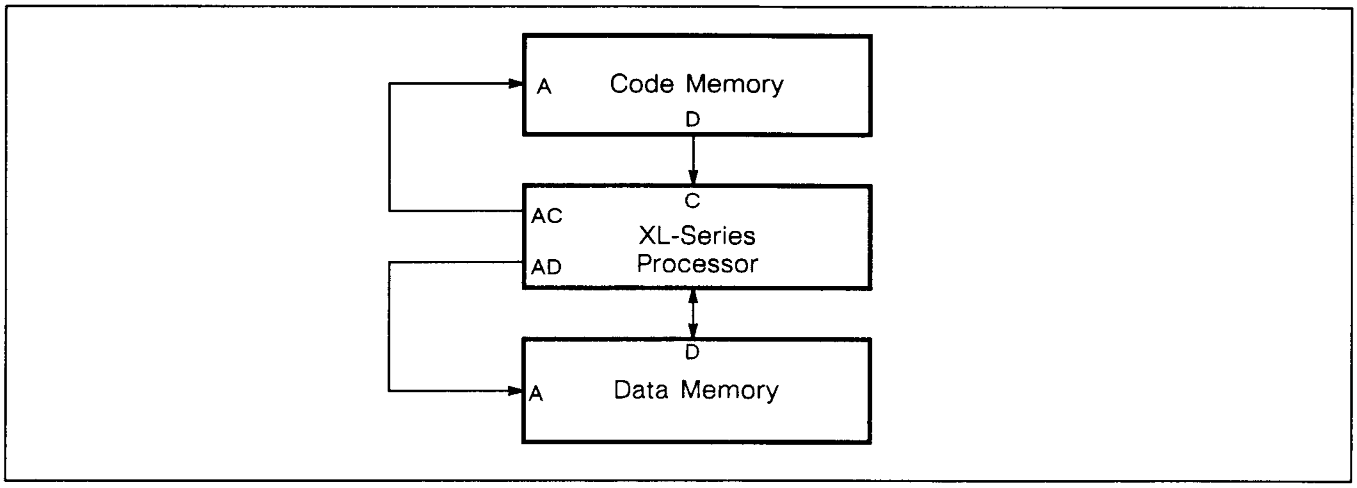


Figure 1. Simplified block diagram of an XL-series processor

The XL-Series Processor Family

XL-8000 PROCESSOR

The XL-8000 is a general-purpose 32-bit integer processor with enhancements to support high-speed bit-manipulation, address generation, and arithmetic. It achieves a sustained processing rate of 8 MIPS (millions of instructions per second), with a peak of 12.5 MIPS.

The XL-8000 is useful in applications that require high-speed integer processing, such as 2-D graphics, logic simulation, communications, and control.

XL-8032 PROCESSOR

The XL-8032 is a 32-bit floating point processor that achieves a sustained processing rate of 8 MIPS and 5 MFLOPS (millions of floating point operations per second) in the 32-bit IEEE format, with a peak floating point rate of 25 MFLOPS. It has the same integer instruction set as the XL-8000.

The XL-8032 is a cost-effective processor for applications that need fast single-precision floating point, such as graphics transformation or digital signal processing.

XL-8064 PROCESSOR

The XL-8064 is a 64-bit floating point processor that achieves a sustained processing rate of 7 MIPS and 6 MFLOPS in either single- or double–precision IEEE floating point formats, with a peak floating point rate of 20 MFLOPS. The floating point unit is a full implementation of the IEEE floating point standard. The XL-8064 has the same integer instruction set as the XL-8000.

The XL-8064 is ideal for applications that need double-precision floating point, such as circuit simulation and general-purpose scientific computing.

Feature	**XL-8000**	**XL-8032**	**XL-8064**
Floating point Capability	software	32-bit IEEE-format	32- or 64-bit Full IEEE implementation
Code Bus	32 bits	64 bits	64 bits
Data Bus	32 bits	32 bits	32 or 64 bits
Speeds	80, 100, 120 ns	80, 100, 120 ns	100, 120 ns
Peak MIPS	12.5	12.5	10
Sustained MIPS*	8	8	7
Peak MFLOPS	-	25	20
Sustained MFLOPS	-	5	6
Number of VLSI components	2	3	3
Maximum Code Bandwidth	50 MB/sec	100 MB/sec	80 MB/sec
Maximum Data Bandwidth	50 MB/sec	50 MB/sec	80 MB/sec

* Sustained MIPS give performance relative to a VAX 11/780, which has a sustained performance of 1 MIP.
All performance numbers are for the fastest speed grade.

Figure 2. Comparison of the XL-Series processors

Related Documents

XL-SERIES COMPONENT DATA SHEETS

Data sheets for the XL-8136 program sequencing unit, the XL-8137 integer processing unit, and the XL-3132 floating point unit.

XL-SERIES PROGRAMMER'S BINDER

This binder contains software and programming information, including discussions of software tools, programming techniques, compilers, and the XL-Series instruction set.

XL-SERIES SYSTEM DESIGNER'S BINDER

This binder contains information about XL-Series hardware design, systems software, functional simulators, and porting the XL software to a target system.

XL-Series Software

COMPILERS

The XL-Series compilers are advanced optimizing compilers for C and FORTRAN 77. Each is compatible with an industry-standard version of the language—the C compiler is compatible with the UNIX™ portable C compiler, and the FORTRAN compiler conforms to the ANSI FORTRAN 77 standard.

These compilers use a variety of techniques to increase the speed and reduce the size of the program, including automatic allocation of register variables, loop rotation, strength reduction, register coalescing, and static address elimination.

PARALLELIZER

The XL-Series parallelizer takes the output of the compiler and performs a series of optimizations that take advantage of the XL-Series architecture. It places instructions in parallel when possible, makes use of shadow instructions, and takes advantage of the floating point processor's pipelines. The output of the parallelizer is XL-Series assembly code.

The parallelizer recognizes the capabilities of each of the XL-series processors. For example, it converts double-precision floating point instructions to single-precision for the XL-8032, which has no double-precision floating point. For the XL-8000, which has no floating point processor, the parallelizer replaces floating point instructions with calls to routines in a software floating point library.

ASSEMBLER

The XL-Series assembler converts assembly-language instructions into an object module. The assembly language allows exact specification of what happens in each cycle. Speed-critical routines or entire applications can be written in assembly language for maximum performance. Routines written in assembly language and high-level languages can be mixed freely within an application.

LINKER AND LIBRARIAN

The XL-Series linker joins multiple object files into a single executable file. It allows modules compiled at different times to be joined together, and can also link assembly-language modules with compiled modules. The starting addresses of the code and data segments can be specified, and the linker can produce ROM-able code.

The librarian allows a set of modules to be combined into a single file, from which the desired routines can be extracted by the linker.

SOFTWARE SIMULATOR

The XL-Series software simulator is a program that allows applications to be tested in the absence of a working XL system. Programs can be loaded, executed, and debugged on the simulator.

XL-Series Software, continued

FUNCTIONAL SIMULATORS

The functional simulators model the behavior of the XL-Series devices, giving the logic levels on each pin at four points during every clock cycle (before and after the rising and falling edges of the clock). Each simulator models the performance of one XL component, and can be integrated into architectural or timing simulators. The simulators are routines written in C, and are used with the designer's simulation routines to simulate XL hardware designs, and to analyze the behavior of the XL components.

XL-Series Development System

The XL-Series Development System consists of software and the XL-Series development board, which plugs into an IBM PC/AT. Programs written for the XL-Series processors can be run on the board in a UNIX-like environment. The board uses the PC/AT host for console and file I/O, but uses its own 4 MB memory space for code and data.

All XL-Series software runs on the PC/AT under the XENIX operating system.

Application Programmer's Description

The rest of this document describes the XL series from three points of view: that of the applications programmer, that of the system programmer, and that of the hardware designer.

RISC ARCHITECTURE

The XL-Series processors are true 32-bit processors that use a RISC (Reduced Instruction Set Computer) architecture—though they have too large an instruction set to be true RISC machines. They have the following in common with other RISC machines:

Register-to-register, three-address instructions. Both integer and floating point instructions are register-to-register instructions, where two source registers and one destination register can be specified in a single instruction.

Load-store architecture. Accompanying the register-to-register concept is the idea that memory accesses are simple load or store instructions. Memory accesses are broken down into two instructions: address generation and data transfer. Address generation and data transfer instructions can be overlapped to achieve one load per cycle.

Single-cycle execution. All integer instructions except multiply and divide complete in a single cycle. Floating point instructions (except for floating point divide) take no more than four cycles.

Pipelined and parallel execution. The floating point units are pipelined to allow a new operation to be started on every cycle. In addition, the functional units in the XL processors operate in parallel, allowing floating point, integer, memory, and control operations to occur simultaneously.

Large register file. There are 32 general-purpose integer data registers. Large orthogonal register files allow memory accesses to be reduced by maintaining variables and passing parameters in registers instead of RAM. The XL-8032 and XL-8064 each have a 32-element floating point register file as well.

MEMORY ARCHITECTURE

Word-oriented architecture. The 32-bit word is the basic data type of the machine, and all integer instructions produce results of that size. Manipulation of fields smaller than a word must be done in registers. Figure 3 shows how memory is addressed and how bytes are ordered within a word.

PARALLELISM

There are three fields in the instruction word, called the *sequencer field,* the *integer field,* and the *floating point field.* The sequencer field is eight bits wide, the integer field is 24 bits wide, and the floating point field is 32 bits wide (for a total of 64 bits). Most register-to-register operations use only the integer field, most flow-of-control instructions use only the sequencer field, and a few instructions use both fields. Floating point arithmetic instructions occupy the floating point field. Floating point loads and stores require the use of the sequencer field. The XL-8000 doesn't have a floating point field, and so its code word is only 32 bits wide.

In general, any instructions that don't have overlapping fields and don't cause resource conflicts can execute in parallel (resource conflicts occur when two instructions try to use the same bus or register in conflicting ways). Thus a short branch instruction can occur in the same cycle as a bitwise merge instruction, since the short branch uses only the sequencer field, and the bitwise merge uses only the integer field. Conditional branches use the condition code generated by the operation in the integer field to determine whether to branch or not. This allows the test and the branch to occur in the same cycle.

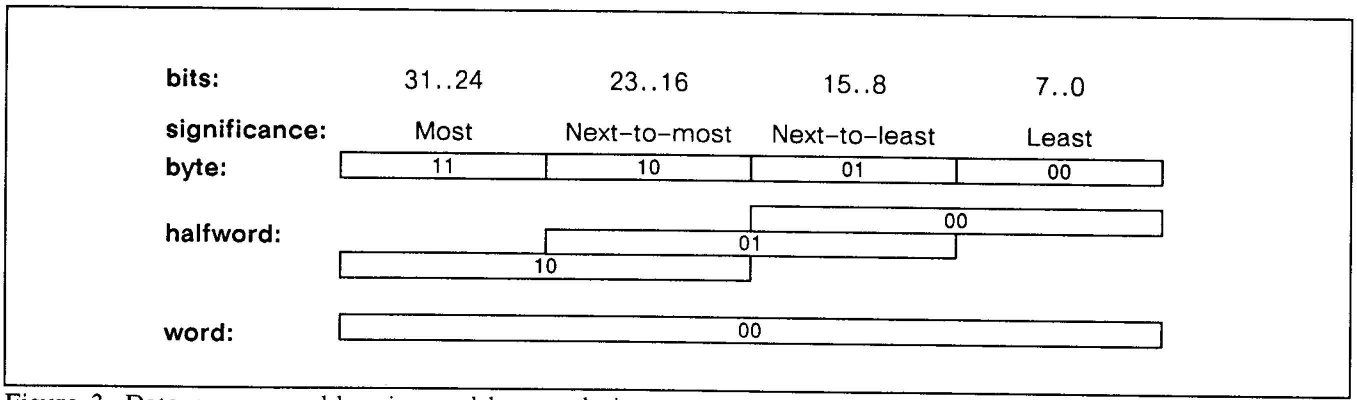


Figure 3. Data memory addressing and byte ordering

Registers

DATA REGISTERS

There are thirty-two 32-bit integer data registers, numbered .r0–.r31. These are general-purpose data registers, any of which can be used as the source or destination for integer register-to-register operations.

FLOATING POINT REGISTERS

The XL-8032 and XL-8064 have thirty-two floating point data registers, numbered .f0-.f31. In the XL-8032, these registers are 32 bits wide. In the XL-8064, they are 64 bits wide. The floating point registers are general-purpose data registers, any of which can be used as the source or destination for any floating point operation.

PRODUCT REGISTERS

The two 32-bit *product registers*, .am and .al, are used by the multiply, divide, and bitwise merge instructions.

FIELD LENGTH REGISTER

The five-bit *field length register* is used by the dynamic bit-manipulation commands (extract, deposit, and merge) to specify the length of the field to be operated on.

SHIFT AMOUNT REGISTER

Like the field length register, the *shift amount register* is a five-bit register used by the dynamic bit-manipulation commands. It specifies the amount of shifting (0–31 bits) to be applied to the desired bit field.

CARRY BIT

The *carry bit* contains the carry from the last arithmetic operation that generated a carry.

STACK

The processor has a 33-word by 32-bit stack for loop counts, branch addresses, subroutine return addresses, and data transfers. The stack consists of a 32-bit *top-of-stack register* and a 32-word by 32-bit RAM. Overflow and underflow trap handlers allow the stack to be extended to arbitrary size in data memory.

Instruction Set

ARITHMETIC FUNCTIONS

The arithmetic instructions consist of signed and unsigned addition and subtraction, with and without carry. One of the operands can be a five-bit immediate instead of a register.

```
add .r0,.r1,.r2          #  Add .r0 to .r1, store
                         # result to .r2
add i.r30,9,.r26         #  Add a 5-bit immediate
                         # to .r30, store result
                         # to .r26
subc .r1,.r30,.r12       #  Subtract .r30 from .r1
                         # with carry, store result
                         # to .r12
```

MULTIPLY AND DIVIDE

A 32-bit signed multiply is performed in eight cycles; a 64/32 bit mixed-precision unsigned divide is done in twenty cycles. Multiplication gives a 64-bit product; division gives a 32-bit quotient and 32-bit remainder.

A multiply or divide can be done in parallel with any operations (except instructions that use the .am or .al registers).

```
# Example of Multiplication
mpy .r1,.r2     #  Multiply regs .r1 and .r2.
nop             #  Wait for result.
nop             #  Useful work could be
nop             # done here instead of
nop             # no-ops.
nop
mov .al, .r3    #  Retrieve low-order 32 bits.
mov .am, .r4    #  High-order 32 bits
```

FLOATING POINT ARITHMETIC

The XL-8032 and XL-8064 have floating point arithmetic instructions, including addition, subtraction, multiplication, integer to floating point conversion, floating point to integer conversion, etc.. The XL-8032 uses the divide lookup table instruction (flut) and a Newton-Raphson approximation to perform floating point division. The XL-8064 performs floating point division in hardware.

Instruction Set, continued

Floating point instructions (except division) take 3 clock cycles to complete on the XL-8032, and 2 cycles to complete on the XL-8064.

Floating point operations can be overlapped (pipelined). A new floating point operation can be started in every cycle, without waiting for the operations in progress to complete. Loads and stores to the floating point unit can occur in parallel with floating point arithmetic.

```
# Pipelined multiplies on the XL-8032.
# One floating point operation can be
# started in every cycle, and loads
# and stores can occur in parallel
fmul .f0, .f1, .f2      ; fload .f9
fmul .f3, .f4, .f5      ; fload .f10
fmul .f6, .f7, .f8      ; fload .f12
fmul .f9, .f10, .f11    ; fload .f13
fmul .f12, .f13, .f14   ; fstore .f2
```

LOGICAL FUNCTIONS

The processor performs the complete set of sixteen bitwise logical operations, including and, or, xor, not, nand, etc.

EXTRACT/DEPOSIT OPERATIONS

The processor has a 32-bit field shifter that can perform field extract, merge, and insert operations in a single cycle. Deposit takes fields aligned at bit zero and converts them to unaligned fields; Extract takes unaligned fields and converts them to aligned fields. Deposit operations fill the bits outside the field with zeros. Extract operations can either zero-extend or sign-extend the extracted value. Merge operations merge the bit field into the target register, leaving bits outside the field unmodified. These instructions are illustrated in figure 4.

The basic forms of these instructions use immediate values for the field length and shift amount parameters. Dynamic extract, deposit, and merge use the values in the shift amount and field length registers. This gives greater flexibility, but generally takes three cycles per operation instead of one, since the shift amount and field length registers must be set up with mov instructions.

```
# Extract
ext  .r15,3,12,.r2
# Typical dynamic extract
mov .r0, .sar            # Set shift amount
mov .r1, .flr            # Set field length
ext  .r15, .sar, .flr, .r3# Do the extract
```

Simple left and right shifts are done with the deposit and extract commands, respectively. Rotates can be done in two cycles with a combination of two field operations.

PRIORITY ENCODE (FIND FIRST ONE)

This instruction counts the number of zero bits that precede the most-significant one bit in a register. This can be used in applications where data is bit-encoded in order of priority.

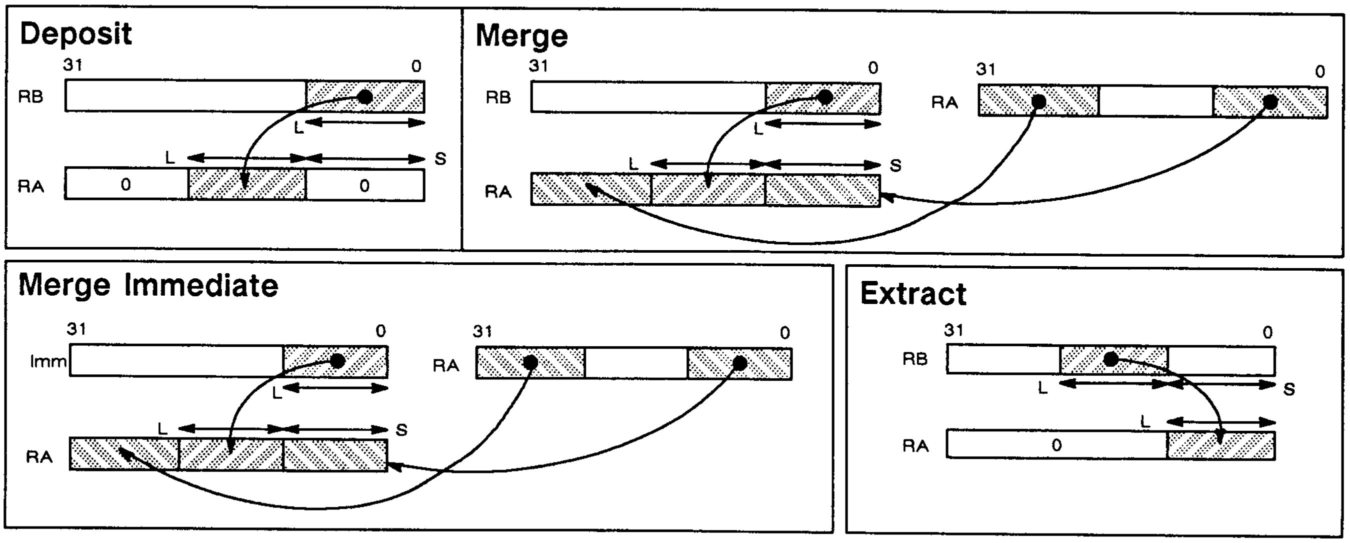
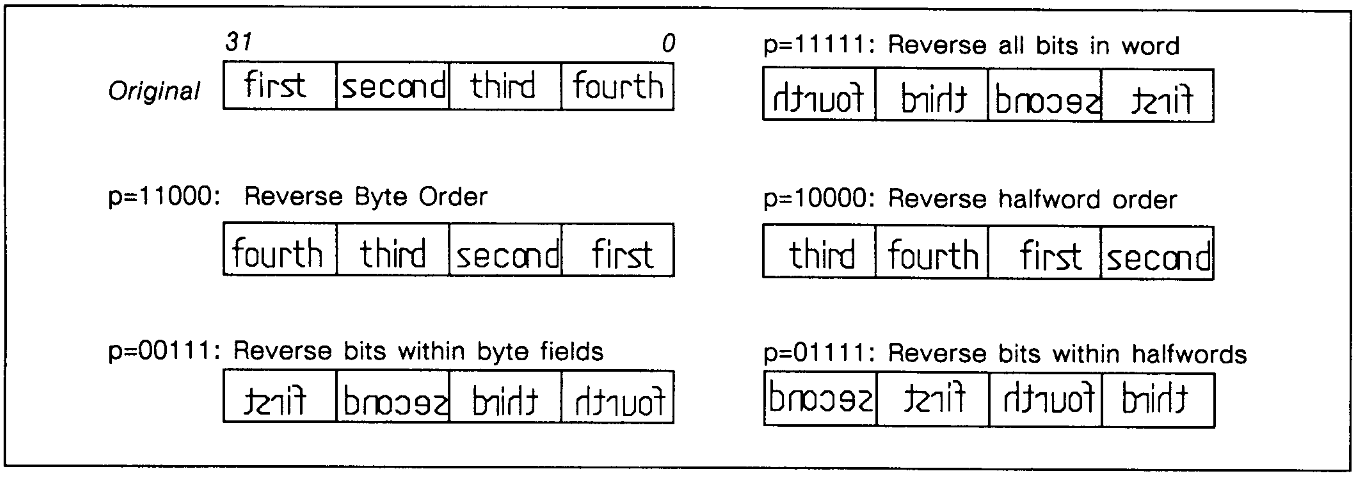


Figure 4. Deposit, extract, and merge instructions

Instruction Set, continued

PERFECT EXCHANGE

This instruction is used to swap fields or reverse the bit order on 2, 4, 8, 16, or 32-bit fields. One use of bit reversal is to calculate addresses in Fast Fourier Transforms. The perfect exchange operation is controlled by a 5-bit *p* field in the instruction. See figure 4.



31
0
Original
first
second
third
fourth
p=11111: Reverse all bits in word
p=11000: Reverse Byte Order
p=10000: Reverse halfword order
p=00111: Reverse bits within byte fields
p=01111: Reverse bits within halfwords

Figure 5. Perfect exchange

Memory Access Instructions

ADDRESS GENERATION

The address generation instructions provide the following addressing forms: base, base plus displacement, base plus index, and base plus scaled index. All of these exist in both pre-modified and post-modified forms. Address generation instructions take a signed value from an immediate field or register, shift it left by 0–3 bits, and add it to a base register—optionally writing the result to another register. The address may be either the result of the addition or the contents of the base register before the addition.

```
# .byte, .half, and .word correspond to a shift of
# 0, 1, and 2 bits, respectively.
addr .r20,.r30.word       #  Basic addressing inst:
                          # add and drive the sum
                          # onto the AD bus.
+addr .r20,.r30.byte,.r10 #  As above, but also
                          # store the sum in .r10
addr+ .r20,.r30,.byte,.r10 #  Drive .r20 to AD bus,
                          # then do the add.  Store
                          # the sum in .r10.
```

LOAD

This instruction places a word from memory into the specified register. The load must be preceded by an addressing instruction. Another operation can be done in parallel with load, so long as it doesn't also try to access memory or the register being loaded.

```
# Load example
addr .r14, 0, .word     # generate the address
subi 15, .r0, .r1 ; load .r23 # Do a subtract,
                        # while at the same time
                        # loading data into .r23.
```

Byte Align For Load. The Processor always loads 32-bit words. Align takes a smaller field, such as a byte, and aligns (and optionally sign-extends) it to fill the entire word. The two-bit *size* field determines the number of bytes to read, and the .adr register is used to determine alignment. Note that this instruction is a register-to-register instruction, and doesn't do the actual loading.

```
# Byte align example
addr .r14, 0, .byte     # generate the address
load .r0                # load the data
align .r0, .r1, .byte   # Take a byte from .r0,
                        # align and store to .r1
```

Instruction Set, continued

The addressing instruction and the align instruction must agree on the size of the data being loaded.

Floating Point Loads. Floating point loads work exactly like integer loads, but the data goes to the register file on the floating point unit instead of the integer unit.

STORE

The store instruction stores the result of the current processor operation to memory. A store instruction must be preceded by an addressing instruction. This instruction always stores a full 32-bit word.

Byte Align And Store. This instruction takes a single field of 1–4 bytes from a register and stores it to memory, optionally checking for sign bit overflow.

Floating Point Stores. Floating point stores take a register in the floating point unit and store it to memory. Floating point stores can occur in parallel with floating point arithmetic operations.

```
# Example of store
addr .r14, 0, .word         #  Generate the address
subc .r1, .r2, .r3 ; store  #  Do a subtract,
                            # and store the result to
                            # memory at the address
                            # in register .r14
```

Program Control Instructions

Program control instructions come in two formats: short and long. Short control instructions use the 8-bit sequencer field. Short instructions include neutralization control, short branches, and some loop control instructions.

Long control instructions use both the integer field and 24-bit sequencer field. These instructions are used for long branches, subroutine calls, register transfer, and miscellaneous operations.

BRANCH INSTRUCTIONS

All branch instructions are relative to the current instruction.

Br (branch) is a long-format instruction that specifies a 24-bit displacement relative to the current instruction.

Shbr (short branch) is a short-format instruction that branches forwards or backwards in the range of –16 to +15 instructions. An integer operation can be performed in parallel with the branch.

Brc (conditional branch) is a short-format instruction that branches conditionally if the parallel integer operation satisfies the test condition. Its range is 0–31 instructions.

Fbr (floating point branch) is like brc, but uses the floating point condition to determine whether to branch or not.

Brstkp (branch to stack and pop) branches by the displacement on the top of stack, then discards the top-of-stack value. Brstkp is a short-format instruction.

CALL AND RETURN INSTRUCTIONS

Bsr (branch to subroutine) pushes the address of the next instruction on the stack and performs a signed 28-bit-displacement branch. It is a long-format instruction.

Bsrstk (bsr to stack) is used for dynamic subroutine calls. It branches to the absolute address on the top of stack. The address of the next instruction replaces the value on the top of stack. It is a short-format instruction.

Rts (return from subroutine) uses the top of stack as an absolute jump address. The value is then popped off the stack. It is a short-format instruction.

LOOPING INSTRUCTIONS

Loop pushes the next instruction address onto the top of stack. It is a short-format instruction.

Endloop is a short-format instruction that branches to the address on the top of stack if the parallel integer operation satisfies the test condition. Otherwise, the address is popped off the stack and the loop is exited. The address is typically put there by a loop instruction.

Fndloop is like endloop, but uses the floating point condition to determine whether to branch or not.

Program Control Instructions, continued

Sob (subtract one and branch) subtracts one from the top of stack. If the result is non-zero, a branch is made by the displacement given in a 24-bit sign-extended immediate value. If the result is zero, the stack is popped, and normal sequential execution resumes. This is useful at bottoms of loops designed to continue a set number of iterations. It is a long-format instruction.

Shsob (short sob) is similar to sob, but branches are specified with a one-extended (i.e., negative) five-bit immediate, rather than a sign-extended 24-bit immediate. It is a short-format instruction

Brp (branch and pop) Branches by a sign-extended 24-bit immediate displacement. The value on the top of stack is popped off and discarded. Used to exit loops prematurely. Brp is a long-format instruction.

NEUTRALIZATION

One reason for the processor's high speed is that it fetches the next instruction at the same time it executes the current instruction. This means that the next instruction has already been fetched when it becomes time to execute it.

When a branch is executed, however, the processor has the instruction following the branch in its instruction pipeline—not the instruction at the destination address. The instruction that has been fetched is called the "shadow instruction." Fetching the correct instruction 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 processor normally *neutralizes* the cycle following taken branch, call, and return instructions. Neutralization effectively turns an instruction into a no-op.

The processor instruction set also provides three additional instructions: *override neutralization, override neutralization and increment stack pointer,* and *reverse neutralization* (ovneut, ovneuti, and revneut). Efficient code—such as that produced by the XL compilers—makes use of these instructions to selectively execute shadow instructions, saving one clock cycle per branch.

TRAPI INSTRUCTION

The trapi instruction is used with an 11-bit immediate to make system calls. The immediate value is pushed onto the stack, and a software interrupt is generated. The interrupt service routine uses the value on the stack to determine which system call is required. This allows user-mode routines to request supervisor-mode services in an orderly manner.

Parallelism

Since there are three fields in the instruction word, a maximum of three operations may be specified for any one instruction cycle—typically an integer operation, a control operation, and a floating point operation. All three will execute in parallel.

It's possible to have more than three operations in progress during a cycle, however, since the integer multiply/divide unit will work in parallel with other instructions, and several instructions can be in the pipeline of the floating point unit.

```
# Example of parallelism
mpy .r1,.r2    #  Start an integer multiply
fmul .f0,.f1,.f2 #  Start a floating point mpy
fadd .f8,.f9,.f7 #  Start a floating point add
# Now start three operations at once:
add .r25,6,.r2 ; bne LABEL ; fsub .f11,.f12,.f13
```

In the last cycle of this example, there are six operations going on at once: Three floating point operations (a multiply, which finishes in the next cycle, an add, which will finish one cycle later, and a subtract, which will finish two cycles later), an integer multiply (executing in the independent multiply/divide unit), an integer add, and a conditional branch (which tests the result of the integer add).

Systems Programmer's Description

The systems programmer is in control of interrupts, system initialization, privileged instructions, and debugging. The XL-Series processors have a number of features that make systems programming easier, including an extra bank of data registers, a trap that signals imminent stack overflow or underflow, vectored interrupts, individual and master interrupt enables, and a breakpoint/watchpoint register.

SUPERVISOR MODE

The processor enters supervisor mode on reset and when it honors an interrupt. A number of instructions are accessible only from supervisor mode, including most instructions that modify special registers directly (user-mode instructions are also available in supervisor mode). User programs (which in this case means any code that doesn't run in supervisor mode) are expected to request system services by using trapi instructions, which perform software interrupts—and thus put the processor into supervisor mode. On the return from the trap handler, supervisor mode is restored to its previous state.

A privilege violation trap occurs when a user-mode program attempts to execute a supervisor-mode instruction.

Registers

PROGRAM COUNTERS

There are two 32-bit program counters: the *currently executing address* (.cea), the true program counter; and the *currently fetching address* (.cfa), which is the address from which a code word is being fetched for later execution. Branches are taken relative to the currently executing address.

INTERRUPT ADDRESS REGISTERS

There are four interrupt address registers: the *interrupt base register* (.ibr), the *interrupt last address* (.ila), the *interrupt fetch address register* (.ifa) and the *interrupt execute address register* (.iea). The interrupt base register contains the address of the interrupt vector table, which is a 64-word table in code memory. The interrupt last address contains the address of the last instruction that was successfully executed. The interrupt fetch and execute registers hold the old contents of .cea and .cfa during interrupt processing.

ADDRESS HOLDING REGISTER

The processor retains the last address generated by any of the address generation instructions in the .adr register, so it can re-assert the address in the event of an interrupt.

SECOND REGISTER BANK

Registers .r28–.r31 are duplicated in a second bank, which is swapped in and out when the z bit in the processor status register is toggled. These registers (named .r28′–.r31′) are used to save the state of the machine during interrupt processing. The second bank is illustrated in figure 6. Note that .r0–.r27 are always accessible, regardless of the state of the z bit.

STACK POINTER

The five-bit stack pointer (.tos) is a modulo 32 counter which increments on each push and decrements on each pop.

An underflow exception occurs when a pop operation empties the stack. An overflow exception is generated when a push operation nearly fills the stack.

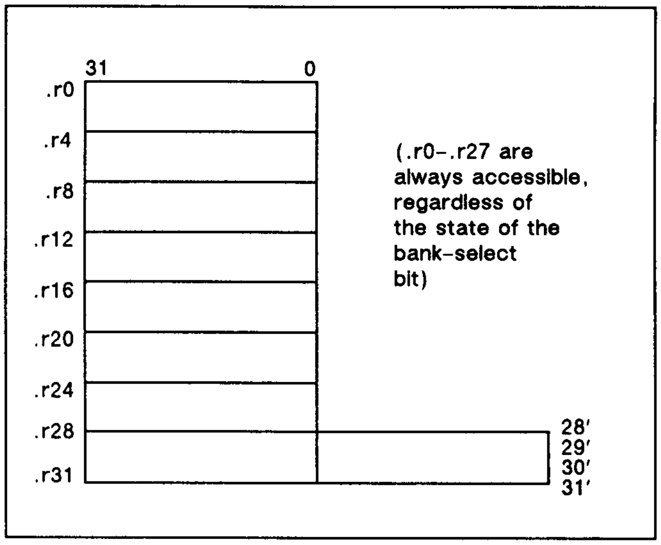

31
0
.r0
.r4
.r8
.r12
.r16
.r20
.r24
.r28
.r31
(.r0–.r27 are always accessible, regardless of the state of the bank-select bit)
28′
29′
30′
31′

Figure 6. Data registers

Registers, continued

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

PROCESSOR STATUS REGISTER

The processor retains some control information in the *processor status register* (.psr). Important fields in the processor status register are the *carry bit* (c), the *field length register* (.flr), which is used in bit-field-manipulation instructions; the *shift amount register* (.sar), which is also used in field-manipulation instructions; and the *register bank toggle* (z). The processor status register is shown in figures 7 and 8.

SEQUENCER STATUS REGISTER

The *sequencer status register* (.ssr) includes the five-bit *top-of-stack register* (.tos), the supervisor mode and branch bits, ten 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 is cleared, all interrupts are disabled.

Instructions that explicitly read or write the sequencer status register are restricted to code running in supervisor mode. The sequencer status register is illustrated in figures 9 and 10.

The flag/enable bits selectively control the interrupts. If an interrupt signal is asserted, or an internal exception occurs, its flag bit is set. If the relevant enable bit is set (and the master interrupt enable bit is also set), then an interrupt sequence also occurs. Interrupt-handling software can examine the flag bits to determine which interrupts have occurred.

The men bit is the master interrupt enable. If men is false and an interrupt occurs, no interrupt routine will be called, but the associated exception flag will still be set.

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

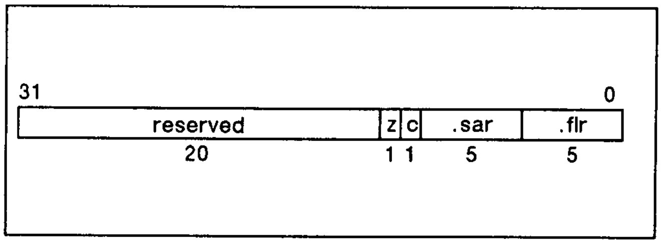


Figure 7. Processor status register (.psr)

Symbol	Meaning
.sar	Shift amount register
.flr	Field length register
z	Register bank select (for .r28-.r31)
c	Carry bit
reserved	Reserved (should be set to zero)

Figure 8. Processor status register bit fields

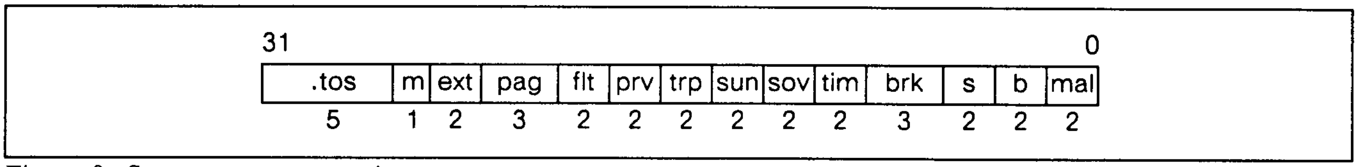


Figure 9. Sequencer status register

Registers, continued

Symbol	Bit #	Name	Meaning
mal	0	*malflg*	flag for misaligned data interrupt
	1	*malen*	enable for misaligned data interrupt
b	2	*b*	reserved. State must be preserved by the programmer
	3	*bi*	reserved. State must be preserved by the programmer
s	4	*s*	reserved. State must be preserved by the programmer
	5	*si*	reserved. State must be preserved by the programmer
brk	6	*brkflg*	flag for breakpoint interrupt
	7	*brkenc*	enable for code breakpoint interrupt
	8	*brkend*	enable for data breakpoint interrupt
tim	9	*timflg*	flag for timer interrupt
	10	*timen*	enable for timer interrupt
sov	11	*sovflg*	flag for sequencer stack overflow interrupt
	12	*soven*	enable for sequencer stack overflow interrupt
sun	13	*sunflg*	flag for sequencer stack underflow interrupt
	14	*sunen*	enable for sequencer stack underflow interrupt
trp	15	*trpflg*	flag for trap instruction interrupt
	16	*trpen*	enable for trap instruction interrupt
prv	17	*prvflg*	flag for privileged instruction interrupt
	18	*prven*	enable for privileged instruction interrupt
flt	19	*ext4flg*	flag for EXT4- interrupt
	20	*ext4en*	enable for EXT4- interrupt
pag	21	*ext23en*	enable for EXT2- and EXT3- interrupts
	22	*ext2flg*	flag for EXT2- interrupt
	23	*ext3flg*	flag for EXT3- interrupt
ext	24	*ext1flg*	flag for external interrupt EXT1-
	25	*ext1en*	enable for EXT1-
m	26	*men*	master interrupt enable
.tos	27-31	*tos*	five bit top of stack pointer

Figure 10. Sequencer Status Register bit fields

TIMER REGISTER AND INTERRUPT

The processor includes a 32-bit timer register. This register is decremented every clock cycle. When the value becomes negative, the timer flag is set and a timer interrupt occurs.

The timer continues to decrement when negative, allowing accurate timing even if the service routine is interrupted or delayed. The timer may only be set or read from supervisor mode.

BREAKPOINT/WATCHPOINT REGISTER

The 32-bit Breakpoint Register (.brk) is used to provide a code breakpoint or data watchpoint for program debugging. Breakpoints and watchpoints are set by loading the register with the address to be monitored, and enabling the breakpoint or watchpoint interrupt enable bit. When the processor accesses the location being monitored, a breakpoint interrupt will occur.

Interrupts

The processor receives interrupts from four external sources: EXT1-, EXT2-, EXT3-, and EXT4-; and generates seven interrupts internally: BRK, TIM, SOV, SUN, TRP, MAL, and PRV. When an interrupt control line or internal condition becomes active, the sequencer sets the corresponding .ssr interrupt flag. If the master interrupt enable bit of the .ssr is set, and the corresponding .ssr interrupt enable is active, the interrupt will be honored, as described below.

There are sixteen interrupt vector addresses, divided into four classes: EXT1-, EXT2- or EXT3-, TRAP, and Others. There is an interrupt vector for every combination of the four vectors.

INTERRUPT CONTROL LINES

The external interrupt sources are: EXT1-, EXT2-, EXT3- and EXT4-. Each has status and interrupt enable bits in the .ssr.

The interrupt mask bit for EXT1- is called ext1en, and its status bit is ext1flg.

EXT2- shares an interrupt enable bit, ext23en, with EXT3-. Its associated status bit is ext2flg.

EXT3- shares an interrupt enable bit, ext23en, with EXT2-. Its status bit is ext3flg.

EXT4- has an enable bit called ext4en. Its status bit is ext4flg. It is used as the floating point exception interrupt on the XL-8032 and XL-8064.

EXCEPTION SOURCES

There are seven internal exception sources: PRV, SOV, SUN, MAL, TRP, TIM and BRK. Each has a status and interrupt enable bit in the .ssr.

PRV is set when an attempt is made to execute a privileged instruction while not in supervisor mode. Its interrupt enable and status bits are prven and prvflg, respectively.

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. SUN occurs when the last valid data is popped off the stack. The enable and status bits for SOV and SUN are soven, sovflg, sunen, and sunflg.

MAL is the misaligned data exception, which occurs when the data to be loaded or stored straddles a word boundary. Its enable and status bits are malen and malflg, respectively. Misaligned loads and stores can be corrected in a trap handler if code memory can be examined by the software. This is possible in some implementations. Otherwise, misaligned loads and stores are unrecoverable errors.

TRP is set by invoking one of the trap instructions. Its enable and status bits are trpen and trpflg, respectively. Trap instructions are software interrupts.

The remaining two exceptions, TIM and BRK, are set on timer interrupts and breakpoints/watchpoints, respectively. Their enable and status bits are timen, timflg, brkenc, brkend (for code and data breakpoints, respectively), and brkflg.

Hardware Designer's Description

PROCESSOR CHIP SETS

All XL-Series processors have an *integer processing unit* (IPU) and a *program sequencing unit* (PSU). Each of these units is a single CMOS device, in a 144-pin PGA package. The XL-8032 has a 32-bit floating point unit, also in a 144-pin PGA package. The XL-8064 has a floating point unit which comes in a 168-pin PGA package.

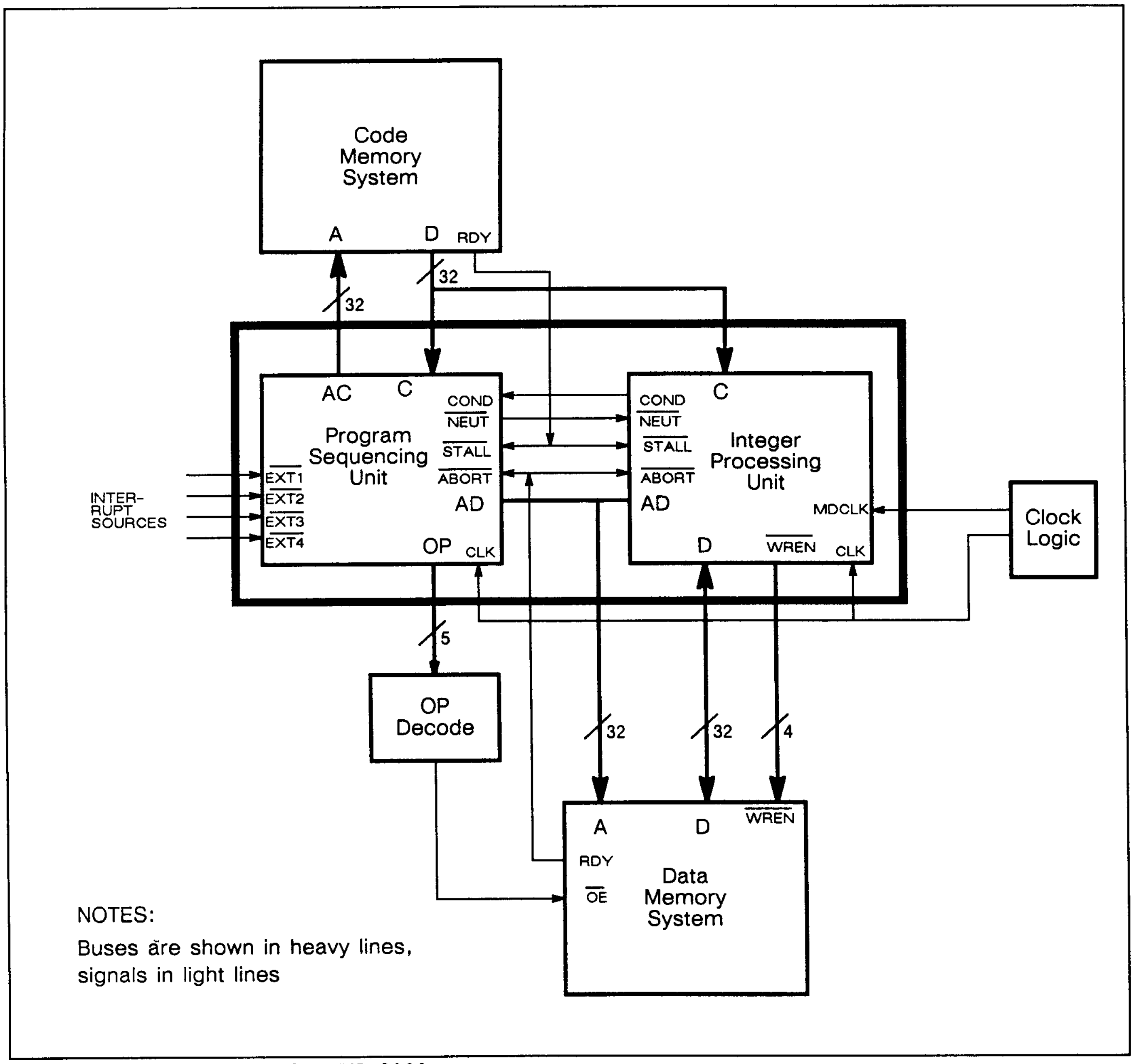
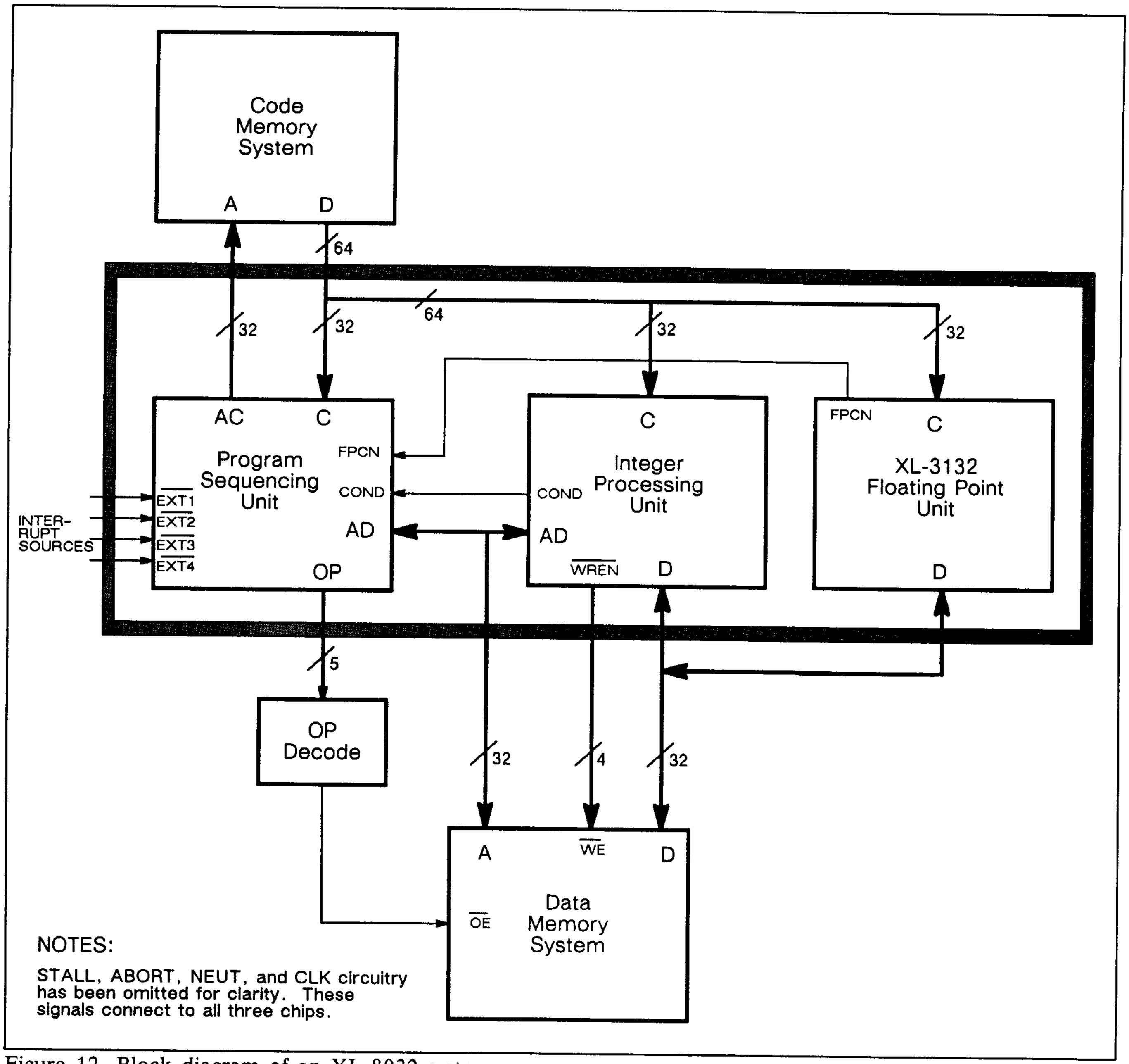


Figure 11. Block diagram of an XL-8000 system

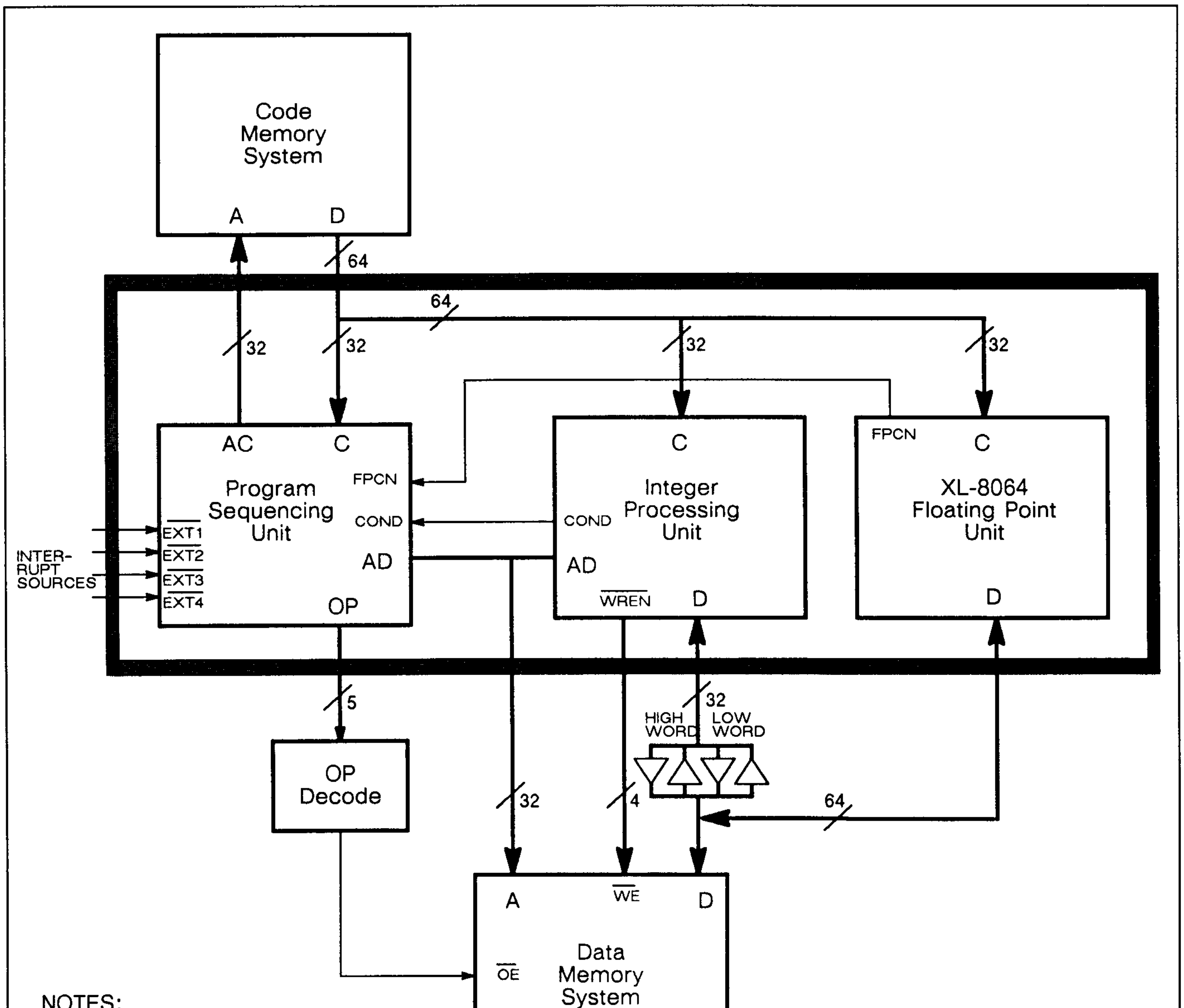


NOTES:

STALL, ABORT, NEUT, and CLK circuitry has been omitted for clarity. These signals connect to all three chips.

Figure 12. Block diagram of an XL-8032 system

Hardware Designer's Description, continued



NOTES:

STALL, ABORT, NEUT, and CLK circuitry has been omitted for clarity. These signals connect to all three chips.

There are both 32- and 64-bit data bus versions of the XL-8064. The 32-bit version has a bus structure identical to the XL-8032.

Figure 13. Block diagram of an XL-8064 system (64-bit data bus version)

Signal Description, Buses

There are two memory buses on the XL processors: one for code and one for data. Having separate code and data space gives the same bus bandwidth as a conventional Von Neumann machine running at twice the clock rate. Another bus, the OP bus, is used both for external I/O and to encode information about the machine state that is useful to the data memory and interrupt systems.

Both code and data buses have 32-bit addresses. The width of the data bus varies among the XL-Series processors. The code word of the XL-8000 is 32 bits wide. The XL-8032 and XL-8064 have 64-bit code words to make room for the floating point instruction field. The XL-8000 and XL-8032 have 32-bit data words, while the XL-8064 has configurations for either a 32- or 64-bit data word. The 64-bit configuration allows double-precision floating point words to be loaded or stored in a single bus transaction.

AC BUS

The $AC_{31..0}$ *Code Address Bus* is driven by the sequencer. It sends a 32-bit instruction address to the code memory. The code address is not latched by the sequencer, so an external address latch is necessary between the AC bus and code memory. The sequencer is the only XL chip that uses the AC bus. The high-order bits of the AC bus can be left floating if they aren't going to be used.

The AC bus produces code word addresses, not byte addresses. The code word size is 32 bits on the XL-8000 and 64 bits on the XL-8032 and XL-8064.

AD BUS

The $AD_{31..0}$ *Data Address Bus* provides addresses for data memory operations. It is also used for intra-processor communication and communication with external hardware. It connects to the integer processor and sequencer, but not to the floating point processor. It can also be used as a bidirectional data bus for transfers to and from other hardware. The AD bus is not latched, so an external address register is necessary between the AD bus and data memory.

All 32 bits of the AD bus need to be attached between the sequencer and integer processor, but the high-order bits can be ignored by external memory if the full 32 bits of memory space isn't going to used.

The address on the AD bus is a byte address. Since the integer processor and floating point unit always load full words, and the write-enable bits (WREN-) indicate which bytes are to be written, it isn't necessary to connect $AD_{1..0}$ to memory or peripherals.

C BUS

The $C_{31..0}$ or $C_{63..0}$ *Code bus* is driven by the code memory with the 32- or 64-bit instruction word. The code word is latched by the processor at the rising edge of the clock. This bus provides instruction words for all chips in the XL-Series chip set.

D BUS

The $D_{31..0}$ or $D_{63..0}$ *Data bus* is used as a bidirectional input/output bus. It transfers data words between data memory and the processor. The integer processing unit always loads 32-bit words, but can store bytes or halfwords. The data is latched by the processor. This bus is connected to the floating point processor and integer processor, but not to the sequencer.

The 64-bit configuration of the D bus can be used by the XL-8064 floating point processor to allow double-precision floating point words to be loaded or stored in one bus transaction. WEITEK also supports a 32-bit bus with the XL-8064. In either case, the integer processor accesses memory 32 bits at a time; only the floating point processor can do 64-bit transfers.

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 decodes 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. The OP bus is on the XL-8136 sequencer.

Signal Description, Bus Control

ABORT–

ABORT– is a "not-ready" line for data memory. It is asserted by the data memory subsystem when the data at the requested address cannot be accessed on the next cycle. The XL chips each cancel both their current and next instructions, and attempt to re-start the current instruction. The instruction will be re-executed the cycle after ABORT– is de-asserted. All the XL-Series chips must have their ABORT– lines tied together.

STALL–

STALL– is a "not-ready" line for code memory. It is asserted 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 attempt to fetch it again on the next cycle. The instruction will be executed when STALL– is de-asserted. All the XL-Series chips must have their STALL– lines tied together.

OEA– OEAD–, OED–, OEX–, AND OEAC–

OEAD–, OED–, and OEAC– are asynchronous output enable signals for the AD, D, and AC buses respectively. The buses are tri-stated when disabled. OEX– is the XL-3132 equivalent for OED–. OEA– is the XL-8137's equivalent for OED–. These signals allow easy access to the code and data buses for cycle-stealing or DMA hardware.

WREN–

The WREN$\bar{3}_{..0}$ signals are write-enables for each byte in the data word. The WREN– lines are driven when a store instruction is executed by the processor.

Other Signals

NEUT–

NEUT– is a signal that goes from the PSU to the IPU and FPU. It is not normally used by hardware outside the processor chip set. NEUT– is asserted by the sequencer, and instructs all XL chips to cancel their current instructions. This is done on 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.

EXT1–, EXT2–, EXT3–, and EXT4–

Level-sensitive interrupt request lines. The current instruction is allowed to complete (unless ABORT– is also asserted, in which case the instruction is canceled, and will be re-executed when the interrupt routine returns), 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 only at the rising edge of the clock.

EXT4– is used by the XL-8032 and XL-8064 as a floating point exception interrupt.

COND

Condition code signal. Goes from the IPU to the PSU. Not normally used outside the processor chip set.

FPCN

Floating point condition code. Goes from the FPU to the PSU. Not normally used outside the processor chip set. Tied to GND in the XL-8000, which does not have a floating point unit.

ZERO

XL-3132 zero condition output. Not used in XL. (Leave floating.)

CLK

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

MDCLK

The multiply/divide clock signal, MDCLK, is a single-phase TTL-level clock signal. This signal must be synchronized to the rising and falling edges of the CLK signal, and runs at twice the frequency of CLK.

SUP

An output that indicates that the processor is in supervisor mode. Can be used to implement protected memory.

Other Signals, Continued

RESET-

A level-sensitive input that resets the sequencer and causes a branch to address 0. The stack pointer is initialized to 31 (empty stack), and supervisor mode is set. The other registers in the chip are undefined. Reset is *not* useful as a non-maskable interrupt.

After the power-up reset, only the program counter and the sequencer status register are defined. All other register contents are undefined. Registers that can cause exceptions (such as the timer and breakpoint registers) must be initialized before their exceptions are enabled.

SMODE

Selects the store mode in the IPU. SMODE is set to 1 in XL systems.

OEA+

A signal on the XL-8137 that causes the ALU result to be driven onto the AD bus on every cycle. Tied low in all XL configurations. Note that this is *not* the same signal as OEA-.

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. Reserved for future expansion.

Code Memory System

In its simplest form, an XL code memory system looks the system shown in figure 14. The code address comes out the AC bus, is latched by a set of 373-type latches, and fed into a 32- or 64-bit-wide array of ROMs or static RAMS. The output of the memory chips is driven onto the C bus.

More complex memory systems—such as cached DRAM or static column DRAM—won't always have code ready at the end of a cycle, so the STALL- line has been provided for memory handshaking. Asserting STALL- will cause the code fetch cycle to be retried on the next clock cycle. STALL- can be asserted for as many cycles as necessary to retrieve the code word.

Memory faults such as accesses to non-existent memory or virtual memory page faults can be corrected by asserting STALL- and an interrupt at the same time. The interrupt takes precedence over the stall, so the interrupt routine can take corrective action and return, and the stalled instruction will be tried again.

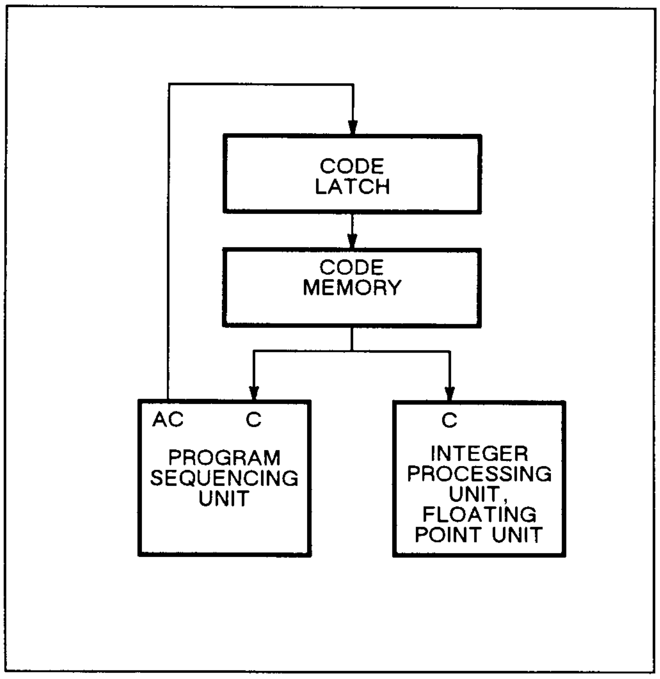


Figure 14. Simple code memory system

Data Memory System

A simple data memory system is shown in figure 15. Data addresses are driven onto the AD bus by the processor, latched by a set of 374-type registers, and fed into a 32-bit-wide array of static RAMs. The output of the RAMs is driven onto the D bus. More complex memory systems include DRAMs with a static RAM cache, and multiple banks of DRAM.

The OP bus is decoded to determine what operations are taking place during the cycle. An address operation, a read, a write, or an address operation plus a read can take place during a single cycle. The decoded OP bus is used to drive the read/write and output enable lines on the data RAMs, and as an input to the clock generation logic, which needs to refrain from clocking the external address register under certain conditions.

If the memory system is not going to be ready in time, ABORT- is used as a data memory handshake signal. ABORT- can be held for any length of time.

Unlike STALL-, ABORT- causes the system to back up and re-execute the aborted instruction when it is released. This allows the failed bus transfer to be re-executed. The external address register should not be clocked during an ABORT- sequence.

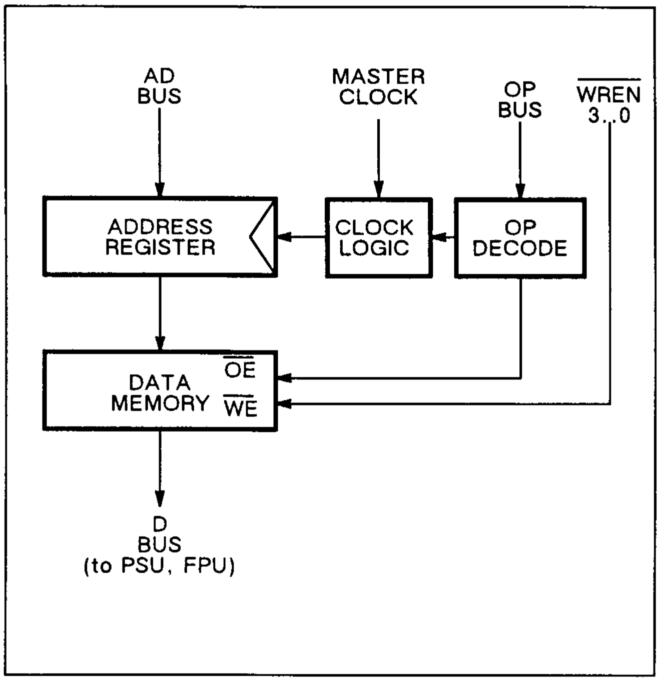


Figure 15. Simple data memory system

OP Output Bus

The OP Bus is available to control external registers and transceivers. Its encoding reports the following actions: address generation, interrupt acknowledge, return from interrupt, data load, data store, and I/O through external registers 0–14.

I/O System

Data can be transferred to external devices over the AD bus at the rate of one 32-bit word per clock cycle. Such transfers are signaled with the OP bus codes 10000–11110, which select "external registers" 0–14. The OP bus code is used to select the external device. Each external register should be associated with one data direction, since there is no data direction signal for OP bus transfers. External I/O is performed in assembly language with the input and output instructions, which transfer one of the 32 data registers over the AD bus.

I/O can also be memory-mapped. External DMA can be performed by tri-stating the buses and asserting ABORT–; at which point data memory can be taken over for any length of time. External code memory access can be performed by asserting STALL– and OEAC–.

Interrupt System

An external interrupt will only be acknowledged if both the master interrupt enable bit and the individual interrupt enable bit are set on the cycle the interrupt is asserted. Interrupts are level-sensitive and synchronous, and are read at the rising edge of the clock.

INTERRUPT SEQUENCE

When an interrupt is detected, the processor decides whether to allow the current instruction to proceed. This decision is based upon the state of the ABORT– signal.

The current instruction is allowed to complete if the ABORT– signal is not asserted; it is canceled otherwise. This allows the system to re-execute the current instruction after returning from the interrupt.

The processor then enters supervisor mode, neutralizes the fetched instruction, saves state information in .iea, .ifa, and .ssr, and branches to the interrupt vector address.

Interrupts can be nested to any depth by saving the processor state and re-enabling 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).

Power-up and Initialization

On power-up, the state of the processor is undefined. RESET– should be held while the the system is powered up, then released. RESET– is a level-sensitive, synchronous signal that is sampled on the rising edge of the clock. When RESET– is asserted, a branch to absolute address zero occurs.

Timing

The figures below give cycle-by-cycle timing for typical bus operations. Both code and data memory systems use overlapped address generation and loads. This memory pipeline allows the system to use slower RAMs without a performance penalty. Loads on both memory systems occur at the rate of one per cycle. Stores on the data memory system can occur at the rate of one every two cycles. I/O transfers using the input and output instructions can take place every cycle.

OP bus values are encoded to show an external register address or a combination of address generation, load/store, interrupt acknowledge, and similar operations. There is also a default encoding that occurs when none of the other conditions apply. These conditions are shown in the OP bus entries in the timing diagrams. The OP bus bit encodings are given in the XL-8136 data sheet.

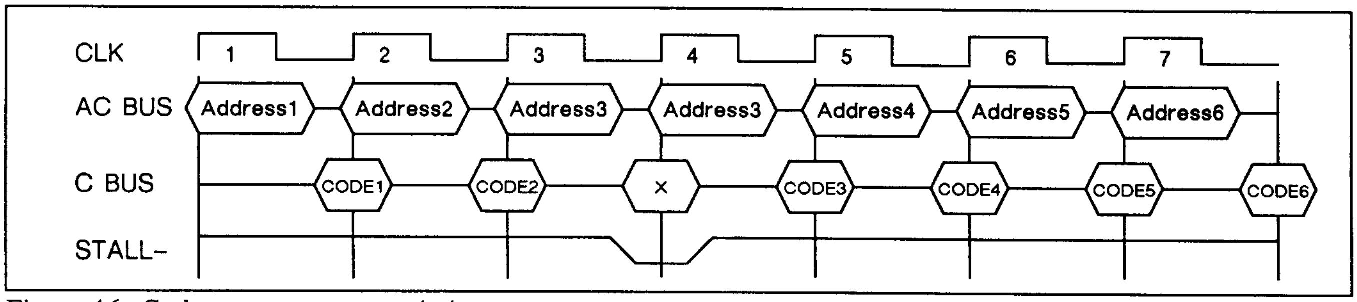


Figure 16. Code memory system timing

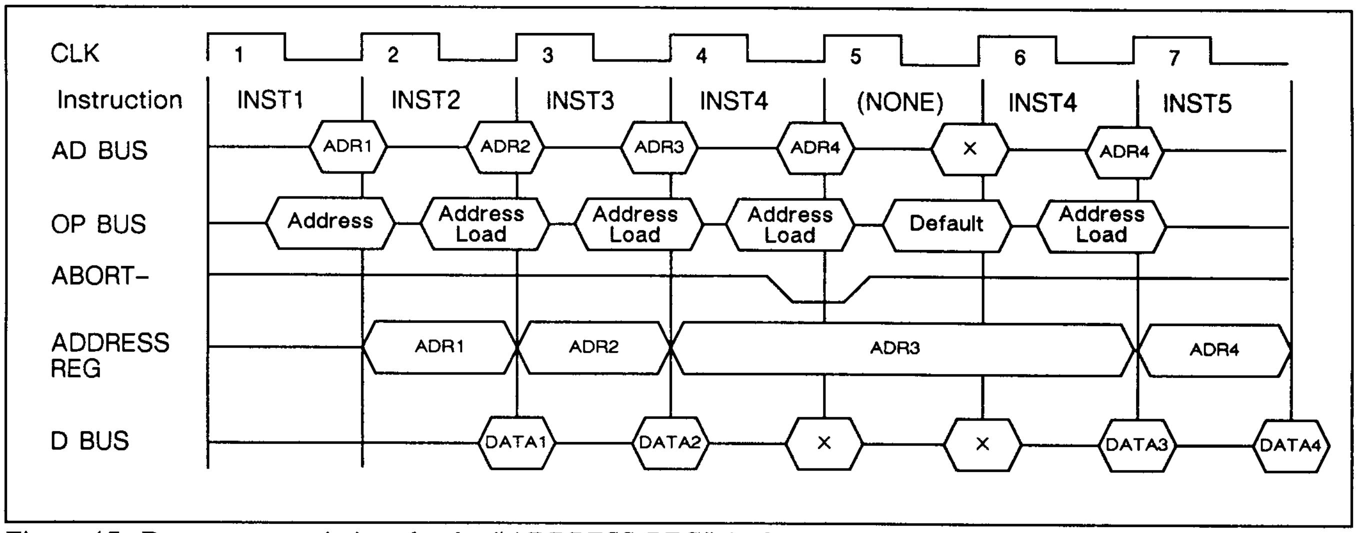


Figure 17. Data memory timing—loads. "ADDRESS REG" is the external data address register

Timing, continued

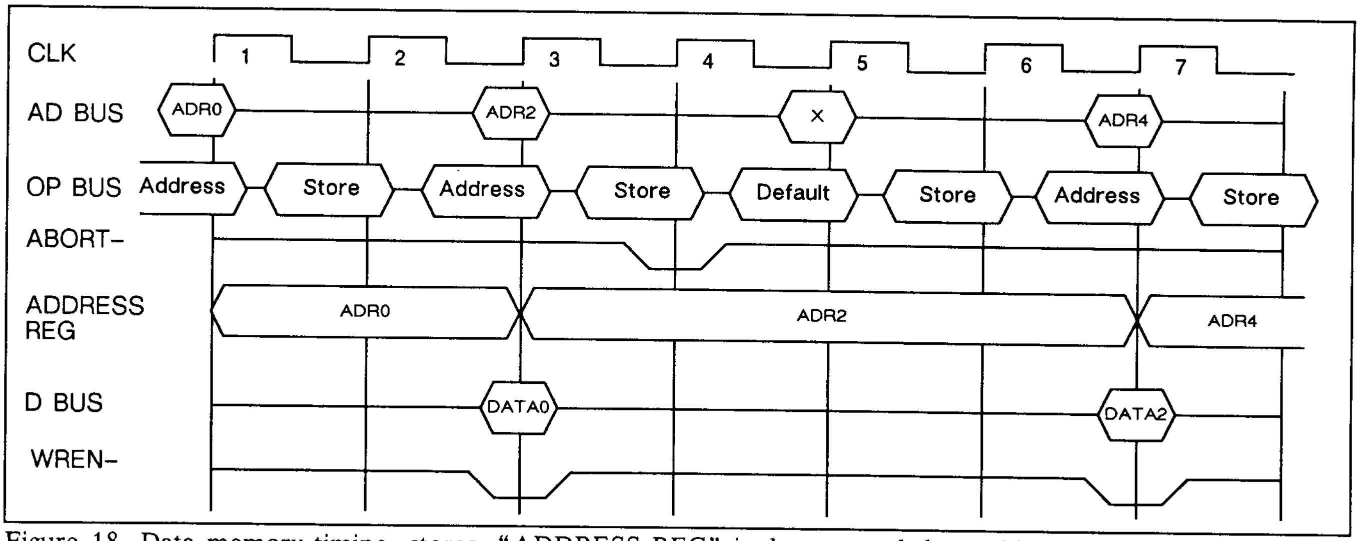


Figure 18. Data memory timing—stores. "ADDRESS REG" is the external data address register

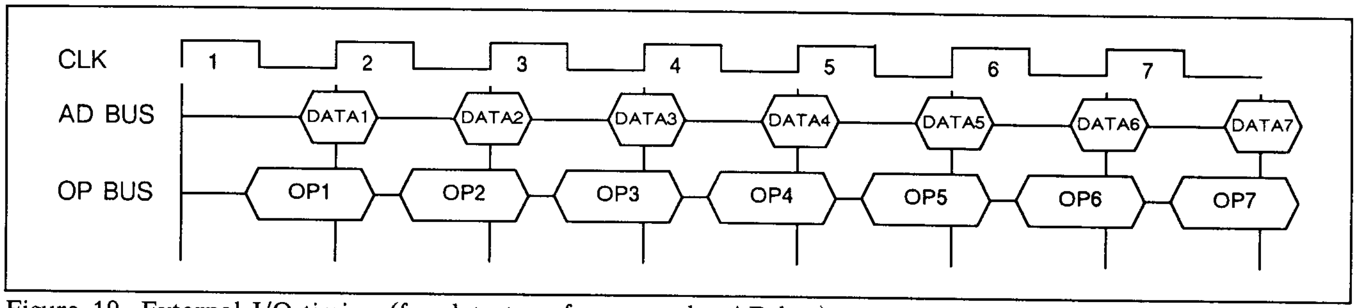


Figure 19. External I/O timing (for data transfers over the AD bus)

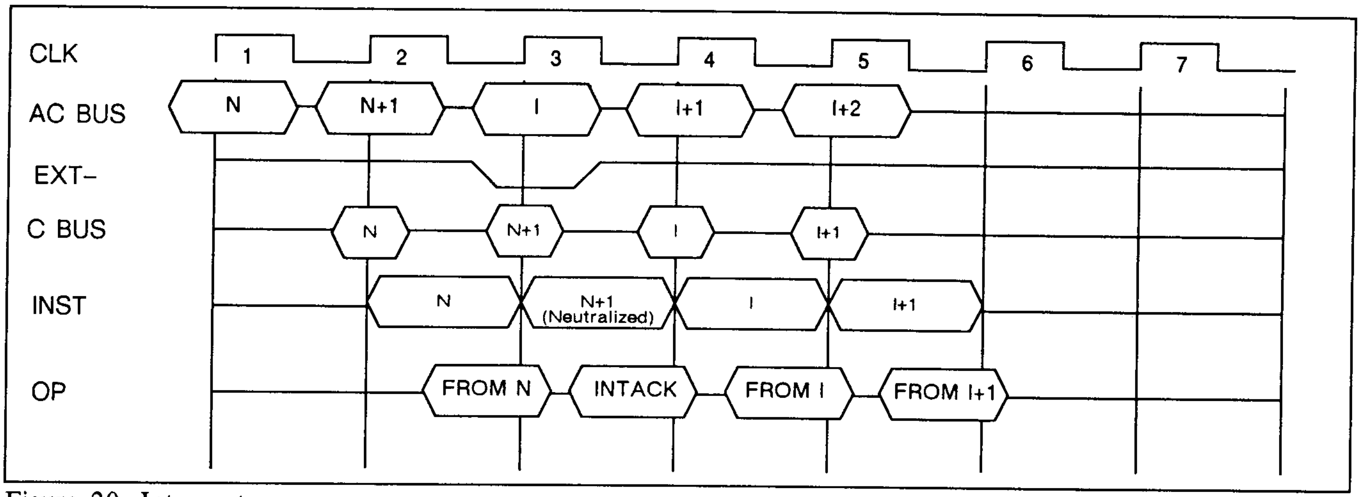


Figure 20. Interrupt sequence

Instruction Set Summary

XL-8000 INTEGER INSTRUCTIONS

add	Signed Add
adda	Add Address
addai	Add Address Immediate
addam	Add Register .am
addami	Add Register .am Immediate
addamis	Add .am Immediate plus Sign
addc	Signed Add with Carry
addi	Signed Add Immediate
addi10	Signed Add 10-bit Immediate
addr	Generate Address (no increment)
addr+	Indexed Addressing (post-increment)
+addr	Indexed Addressing (pre-increment)
addrd	Generate Address with Displacement
addshft	Shift and Add
align	Byte Align for Load
and	Logical AND
asrtadr	Put Address Register on AD Bus
bmerge	Bitwise Merge
clr	Clear
dep	Deposit
div	Divide
ext	Extract
ffo	Find First One (Priority Encode)
input	Input from AD Bus
ldamal	Load Multiply Result Registers
load	Load from Memory
mer	Merge Bit Fields
meri	Merge Immediate
mov	Move
movi	Move Halfword Immediate
movih	Move Immediate High
mpy	Multiply
nand	Logical NAND
neg	Signed Negate
nop	No Operation
nor	Logical NOR
not	Logical NOT
or	Logical Or
output	Output to AD Bus
pexch	Perfect Exchange
rapsr	Restore .adr and .psr
salign	Signed Byte Align
set	Set to Ones
setsema	Set Semaphore
srpsr	Save and Restore .psr
store	Store to Memory
sstore	Signed Store to Memory
sub	Signed Subtract
suba	Subtract Address
subc	Signed Subtract with Carry
subai	Subtract Address Immediate
subi	Signed Subtract Immediate
swap	Save .psr and Swap Banks
uadd	Unsigned Add
uaddc	Unsigned Add with Carry
usub	Unsigned Subtract
usubc	Unsigned Subtract with Carry
xnor	Logical XNOR
xor	Logical XOR

XL-8000 CONTROL INSTRUCTIONS

br	Branch
brp	Branch and Pop
brstkp	Branch to Stack and Pop
bsr	Branch to Subroutine
bsrstk	Branch to Subroutine from Stack
cont	Continue
endloop	Conditional End of Loop
loop	Enter Loop
ovneut	Override Neutralization
ovneuti	Override Neut. and Increment Stack
pops	Pop from Sequencer Stack
pushs	Push onto Sequencer Stack
revneut	Reverse Neutralization
rfi0	Return from Interrupt 0
rfi1	Return from Interrupt 1
rts	Return from Subroutine
seq	Sequencer Housekeeping Instruction
shbr	Branch (short form)
shsob	Sob (short form)
sob	Subtract One and Branch
trap	Trap
trapb	Trap and Back Up
trapi	Trap Immediate

Instruction Set, continued

XL-8032 INSTRUCTION SET

The XL-8032 can execute all of the XL-8000 instructions, and has the following instructions, as well:

fadd	Floating Point Addition
fabs	Floating Point Absolute Value
fbr	Floating Point Branch
fclr	Clear Floating Point Register
fclsr	Clear Floating Point Status Register
fix	Float-to-Fix Conversion
fload	Load Floating Point Data
float	Fixed-to-Float Conversion
flut	Read Floating Point Look-up Table
fmac	Multiply-Accumulate
fmode	Set Floating Point Mode
fmov	Copy Floating Point Register
fmul	Floating Point Multiplication
fstore	Floating Point Store
fstsr	Store Floating Point Status Register
fsub,fsubr	Floating Point Subtraction

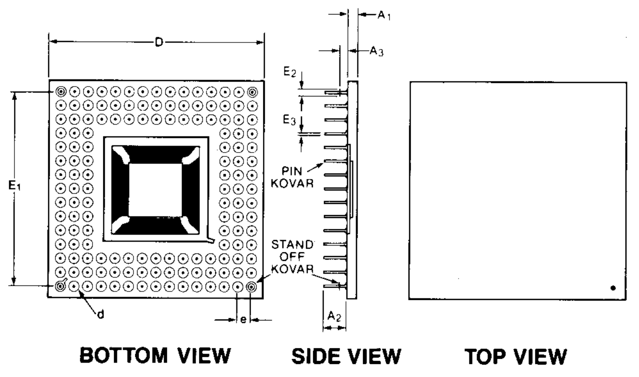
XL-8064 INSTRUCTION SET

The XL-8064 can execute all XL-8000 and XL-8032 instructions, with the exception of the divide look-up table instruction, flut (which isn't necessary on the XL-8064, since it performs division directly).

dfadd	Double Floating Add
dfcnvt	Convert Double to Single
dfdiv	Double Floating Divide
dfix	Double-Precision to Integer (trunc.)
dfixr	Double-Precision to Integer (round)
dfloat	Fixed to Double-Precision
dfmul	Double Floating Multiply
dfsub	Double Floating Subtract
dload	Double-Precision Load
dloadl	Double-Precision Load L.S. Data
dloadm	Double-Precision Load M.S. Data
dstore	Double-Precision Store
dstorel	Double-Precision Store L.S.
dstorem	Double-Precision Store M.S.
fdcnvt	Convert Single to Double
fdiv	Floating Point Divide
fixr	Float-to-Fix Conversion (round)

PRELIMINARY DATA
May 1987

Physical Dimensions



Symbol	LIMITS	
	INCHES	MM
	MIN MAX	MIN MAX
A1	.080 ± .008	2.032 ± 0.203
A2	.180 typ.	4.572 typ.
A3	.050	1.270
D	1.575 sq. ± .016	40.005 sq. ± 0.406
E1	1.400 sq. ± .012	35.560 sq. ± 0.305
E2	.050 dia. typ.	1.270 dia. typ.
E3	.018 ± .002	
d	.070 dia. typ.	1.778 dia. typ.
e	.100 typ.	2.540 typ.

Figure 21. Physical dimensions for all XL-Series devices except the XL-8064 floating point unit.

Ordering Information

DEVICES	PACKAGE TYPE	SPEED	TEMPERATURE RANGE	ORDER NUMBER
2	144-Pin Grid Array	120 ns	T_C = 0–85 °C	XL-8000-120-GCD
2	144-Pin Grid Array	100 ns	T_C = 0–85 °C	XL-8000-100-GCD
2	144-Pin Grid Array	80 ns	T_C = 0–85 °C	XL-8000-080-GCD

Figure 22. Ordering information for the XL-8000

DEVICES	PACKAGE TYPE	SPEED	TEMPERATURE RANGE	ORDER NUMBER
3	144-Pin Grid Array	120 ns	T_C = 0–85 °C	XL-8032-120-GCD
3	144-Pin Grid Array	100 ns	T_C = 0–85 °C	XL-8032-100-GCD
3	144-Pin Grid Array	80 ns	T_C = 0–85 °C	XL-8032-080-GCD

Figure 23. Ordering information for the XL-8032

DEVICES	PACKAGE TYPE	SPEED	TEMPERATURE RANGE	ORDER NUMBER
2 1	144-Pin Grid Array 168-Pin Grid Array	120 ns	T_C = 0–85 °C	XL-8064-120-GCD
2 1	144-Pin Grid Array 168-Pin Grid Array	100 ns	T_C = 0–85 °C	XL-8064-100-GCD

Figure 24. Ordering information for the XL-8064