Chapter 2
L64801 Integer Unit

This chapter provides a description of the L64801 Integer Unit, also referred to as the IU. The topics in this chapter include:

2.1 General Description

The L64801 Integer Unit (IU) is a high-speed implementation of the SPARC 32-bit RISC architecture. LSI Logic fabricates the L64801 IU using LSI Logic's advanced 1.5-micron CMOS process.

The L64801 IU executes instructions at a rate approaching one instruction per processor clock cycle. To achieve this level of performance, the SPARC IU uses Reduced Instruction Set Computer (RISC) architectural principles such as a simple, sufficient instruction set, a large number of on-chip registers, an instruction pipeline, and a closely coupled floating-point coprocessor. The IU is the basic processing engine, and it executes all of the instruction set except for floating-point operations. In addition, the L64801 provides hardware support for multitasking operating systems, and fast interrupt and trap processing. The IU communicates with memory systems via a 32-bit address bus and a 32-bit data/instruction bus.

The IU provides both user and supervisor modes to support a multitasking operating system. Some instructions are privileged and can only be

executed while in supervisor mode. Changing from user to supervisor mode requires taking a hardware interrupt or executing a trap instruction.

The following summary lists the key features of the L64801. The L64801:

- Reduced instruction set computer (RISC) architecture
  - Simple format instructions
  - Most instructions execute in single cycle
- High-performance operation
  - 12 VAX MIPS at 20 MHz
- Large windows register file
  - 120 general-purpose 32-bit registers
  - 7 overlapping windows of 24 windows each
- Hardware pipeline interlocks
- Parallel processing support
- Large virtual address space
  - 32-bit virtual address bus
  - 8-bit address space identifier
- Multitasking support
  - User/supervisor modes
  - Privileged instructions
- High-performance floating-point interface
- Availability in three package types
  - 160-pin Plastic Quad Flat Pack (PQFP)
  - 179-pin Ceramic Pin Grid Array (CPGA)
  - 179-pin Plastic Pin Grid Array (PPGA)

2.2 Internal Registers

The L64801 IU has two types of registers that are visible to software: working registers (R Registers) and control/status registers. Working registers are used for normal operations. The control/status registers control and keep track of the IU's state. This section describes both types of registers.

R Registers

The L64801 IU has 120 general-purpose R Registers. Eight of these registers are Global registers, and the remaining 112 are divided between seven overlapping windows of 24 registers each. Only one window is active at a time. The Current Window Pointer in the Processor State Register selects the active window.

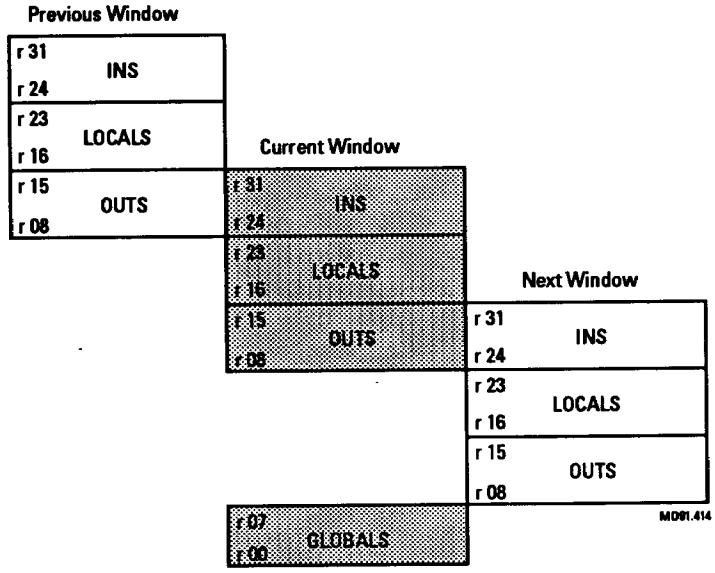
The registers in each window are divided into Ins, Outs, and Locals. Note that the Globals, while not really part of any particular window, can be addressed when any window is active. The registers in the active window and the Global registers are addressed as shown in Table 2.1.

Table 2.1
Register Addressing

Register Numbers	*Name*
R[24] to R[31]	Ins
R[16] to R[23]	Locals
R[8] to R[15]	Outs
R[0] to R[7]	Globals

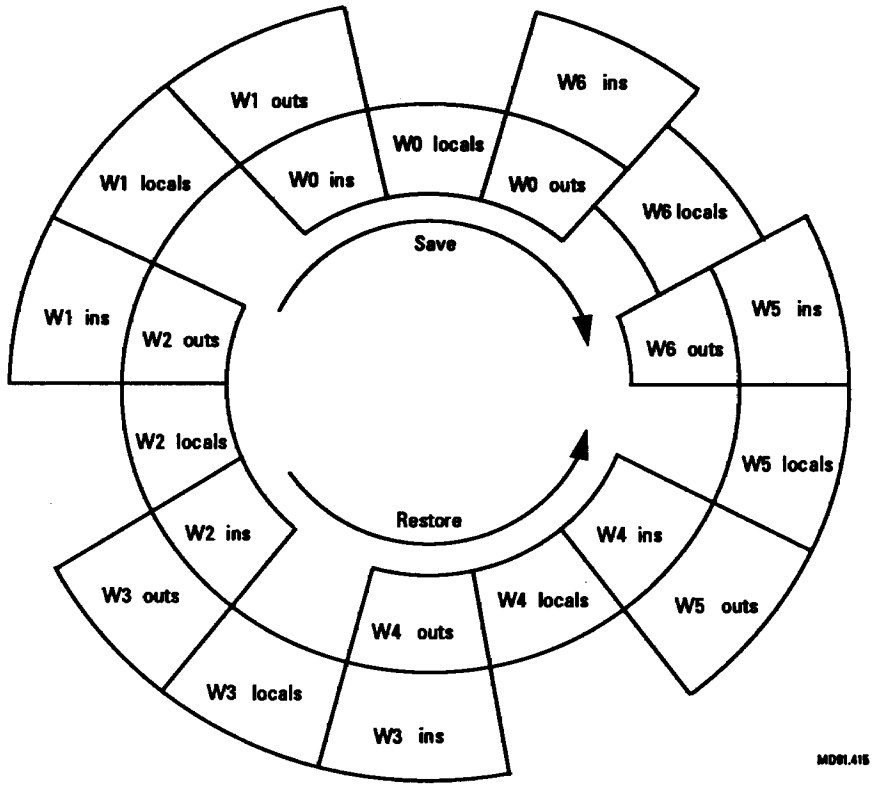
Each window shares its Ins and Outs with adjacent windows. The Ins of the current window are the Outs from the previous window (pointed to by CWP + 1 (modulo 7)). The Outs of the current window are the Ins of the next window (pointed to by CWP – 1 (modulo 7)). The Globals are equally available from all windows, and the Locals are unique to each window. Figure 2.1 shows this overlapping of register windows.

Figure 2.1
Three Overlapping Windows and the Global Registers



The windows are joined together in a circular stack, where the highest numbered window is adjacent to the lowest. The Ins of window 6 are the Outs of window 0. Figure 2.2 shows the relationships between the windows.

Figure 2.2
The Circular Stack of Register Windows



Control/Status Registers

The L64801's control/status registers are all 32-bit read/write registers except as noted below. These registers include the program counters, the Processor State Register (PSR), the Window Invalid Mask Register (WIM), the Trap Base Register (TBR), and the Multiply Step Register (Y).

Program Counters

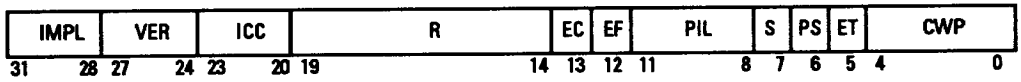
The programmer has access to two program counters, the PC and nPC. The PC contains the address of the instruction currently being executed. The nPC holds the address of the next instruction to be executed (assuming a trap does not occur).

The format of the PC and nPC registers is shown below.

Processor State Register (PSR)

This 32-bit register contains various fields describing the state of the IU. SAVE, RESTORE, Ticc, and RETT instructions modify the register, as do instructions that modify the condition codes. Use the RDPSR and WRPSR instructions to read and write the PSR, respectively.

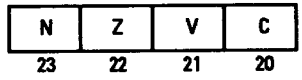
The fields in the PSR are defined as follows:



IMPL **Implementation** **[31:28]**
This field identifies the implementation number of the IU. It is hard-wired and reads as zero.

VER **Version** **[27:24]**
This field identifies the version of this IU implementation. It is hard-wired and reads as zero.

ICC **Integer Condition Codes** **[23:20]**
The ICC field contains the IU's condition codes. The IU sets these bits based on the results of arithmetic and logical instructions whose names end with the letters cc (for example, ANDcc). Use the WRPSR instruction to set these bits. The Bicc and Ticc instructions base their control transfer on these bits, which are defined as follows:



N **Negative** **23**
This bit indicates whether the ALU result was negative for the last instruction that modified the ICC field. A value of one indicates a negative result; zero indicates a non-negative result.

Z **Zero** **22**
This bit indicates whether the ALU result was zero for the last instruction that modified the ICC field. A value of one indicates a result of zero; zero indicates a result that is non-zero.

V **Overflow** **21**

When this bit is one, it indicates that an arithmetic overflow occurred during the last instruction that modified the ICC field. If zero, this bit indicates that an arithmetic overflow did not occur. Logical instructions that modify the ICC field always set the overflow bit to zero.

C **Carry** **20**

When this bit is one, it indicates that either an arithmetic carry out of bit 31 occurred as the result of the last addition that modified the ICC, or that a borrow into bit 31 occurred as the result of the last subtraction that modified the ICC. If zero, this bit indicates that a carry did not occur. Logical instructions that modify the ICC field always set the carry bit to zero.

R **Reserved** **[19:14]**

These bits are reserved. When using the WRPSR instruction, set these bits to zero.

EC **Enable Coprocessor** **13**

Setting this bit to one enables the coprocessor. Setting it to zero disables the coprocessor.

If the coprocessor is disabled or enabled and not present, CPop, CBccc, and coprocessor load/store instructions cause a Coprocessor Disabled Trap.

When the coprocessor is disabled, it retains its state until it is reset or re-enabled. When disabled, the coprocessor can continue to execute any instructions in its queue.

When a coprocessor is present, software uses the EC bit to determine whether or not a process can use the coprocessor. If a process does not use the coprocessor, the coprocessor's registers do not have to be saved when switching contexts.

EF **Enable FPU** **12**

Setting this bit to one enables the FPU. Setting it to zero disables the FPU.

If the FPU is disabled or enabled and not present, FPop, FBfcc, and floating-point load/store instructions cause a Floating-Point Disabled Trap.

When the FPU is disabled, it retains its state until it is reset or re-enabled. When disabled, the FPU can continue to execute any instructions in its queue.

When an FPU is present, software uses the EF bit to determine whether or not a process can use the FPU. If a process does not use the FPU, the FPU's registers do not have to be saved when switching contexts. Also, if the FPU is not present (as indicated by the $\overline{FP}$ signal), the Floating-Point Disabled Trap can emulate the floating-point instruction set.

PIL **Processor Interrupt Level** **[11:8]**

The PIL field defines the minimum IU interrupt level. The IU only accepts interrupts whose level is numerically less than the value in PIL. Since smaller level values equate to higher priority interrupts, software masks lower priority interrupts to the IU by setting the minimum acceptable interrupt level in this field.

Interrupt Level 15 is not masked by PIL and is always accepted when ET is 1.

S **Supervisor** **7**

When set to one, this bit places the IU in supervisor mode. Because instructions that write the PSR require that the IU be in supervisor mode, supervisor mode can only be entered by a software or hardware trap. When S = 0, privileged instructions cause a Privileged Instruction trap.

PS **Previous Supervisor** **6**

This bit indicates the value of the S bit at the time of the most recent trap.

ET **Enable Traps** **5**

When set to one, this bit enables traps. When set to zero, traps are disabled, and all asynchronous traps are ignored. Synchronous traps and Floating-Point/Coprocessor Traps halt the IU and place it in error mode ($\overline{ERROR}$ asserted).

If traps are enabled (ET = 1), take care when you disable them. A RDPSR/WRPSR instruction sequence is interruptible, and it may not be appropriate in some situations. Here are two alternatives: 1) generate a Trap Instruction Trap, which disables traps; or 2) use the RDPSR/WRPSR sequence and write the interrupt trap handlers so that, before returning to the supervisor, they restore the PSR to the value it contained before the trap was taken. The PS bit cannot be restored. In the first alternative, the trap handler should verify that it was called from the supervisor before returning to the supervisor.

CWP **Current Window Pointer** **[4:0]**

The CWP field points to the currently active R Register window. The CWP is decremented (modulo 7) by traps and the SAVE instruction, and it is incremented (modulo 7) by RESTORE and RETT instructions.

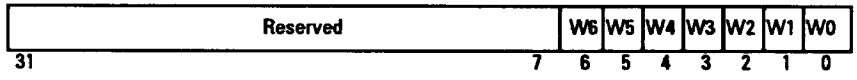
The CWP cannot point to an unimplemented window. Since the L64801 IU provides only seven register windows, bits [4:3] of the CWP always read as zero. The IU ignores attempts to set bits [4:3].

Window Invalid Mask (WIM) Register

The IU uses the WIM register to determine whether a SAVE, RESTORE, or RETT instruction should generate a Window Overflow or Window Underflow Trap. Register bits [6:0] correspond to the seven R Register windows. W0 represents Window 0, W1 represents Window 1, and so on. When software sets one of the bits to one, the IU considers the corresponding window to be invalid. If a SAVE would cause the CWP to point to a window whose corresponding WIM bit equals one, the IU executes a Window Overflow Trap. If a RESTORE or RETT would cause the CWP to point to an invalid window, the IU executes a Window Underflow Trap.

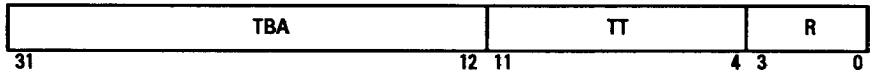
Use the RDWIM and WRWIM instructions to read and write the WIM register, respectively. Bits [31:7] are reserved. The Reserved bits read as zero, and attempts to modify them are ignored.

The WIM register is formatted as shown below.



Trap Base Register (TBR)

The IU uses the three fields of the TBR to generate the address of a trap handler when a trap occurs. After taking a trap and loading the TT field, the IU loads the value in this register into the PC. The TBR fields are defined as shown below.

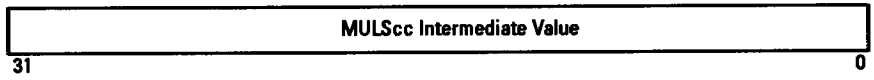


TBA	**Trap Base Address** This field contains the 20 most significant bits of the trap table address. (Note that the Reset Trap is an exception – it traps to address zero.) Software writes this field using the WRTBR instruction.	**[31:12]**
TT	**Trap Type** At the time of a trap, the IU writes the trap type into this field, and it provides an offset into the trap table. The TT field retains its value until a new trap is taken. The WRTBR instruction does not affect this field.	**[11:4]**
R	**Reserved** The least significant four bits of the TBR and the least significant four bits of the trap address are always zero. This feature provides a 4-word spacing between initial trap instructions in the trap table. The WRTBR instruction does not affect this field.	**[3:0]**

Multiply Step Register (Y)

The Multiply Step (Y) register is used with the multiply step instruction (MULScc) to multiply two integers. The integers may be up to 32 bits long and the product up to 64 bits long. At the start of a multiply operation, the IU loads the Y register with the multiplier. To minimize the number of steps needed to multiply, load the shorter of the two integers into the Y register. At the end of multiplication, the Y register contains the least significant bits of the product.

Use the RDY and WRY instructions to read and write the Y register. The register has the following format.



2.3 Instruction Pipeline

This section describes the pipeline of the L64801. The instruction pipeline is depicted in the upper right hand corner of Figure 2.3. The pipeline includes: two instruction buffers (IB1, IB2), three instruction registers (DIR, EIR, and WIR), Instruction Register Decode, and Control Logic.

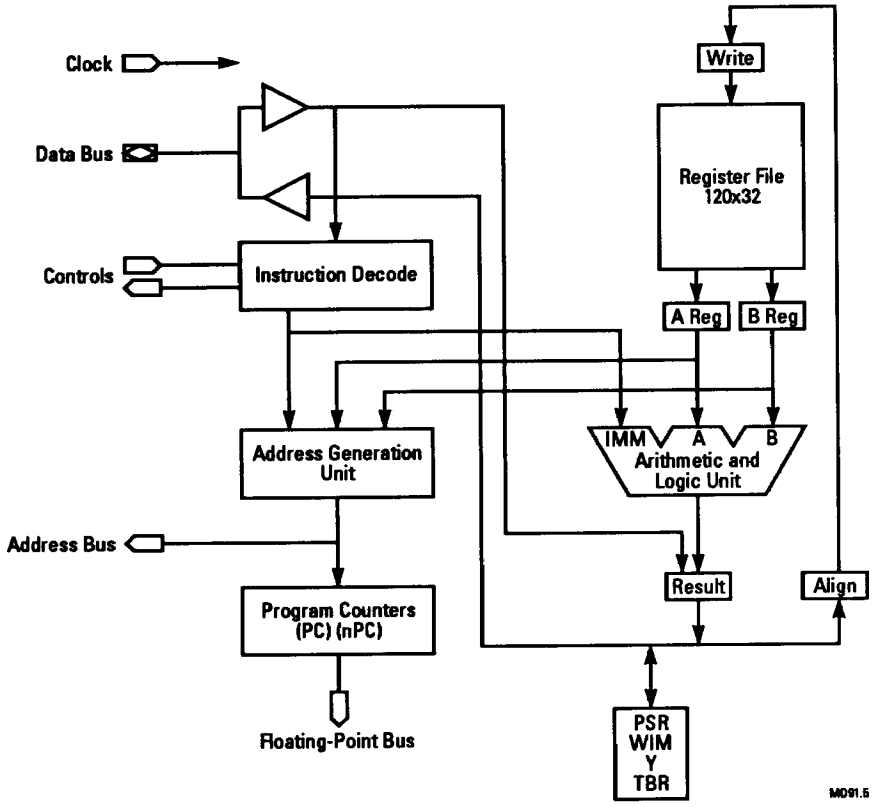
Pipelining is a design technique that improves instruction throughput by overlapping the processing of instructions. A pipelined processor is organized like an assembly line. The processing of instructions is divided into a sequence of steps. Each step is performed in a different pipeline stage. An instruction is passed from stage to stage until all the steps are completed.

The L64801 uses a four-stage pipeline. Most L64801 instructions flow through the pipeline in four cycles from start to finish. During these four cycles, three additional instructions may enter the pipeline. Execution of instructions overlaps, so that effective execution time is a single cycle per instruction. The processor achieves its maximum throughput of one instruction per cycle when a stream of these single-cycle instructions flows through the pipeline.

Some instructions must use part of the processor during more than one cycle. Because these additional cycles cannot be overlapped with other instructions, they increase effective execution time. Instructions that have an effective execution time greater than one cycle are called multiple-cycle instructions.

Memory access instructions comprise the majority of multiple-cycle instructions. All instructions require one Memory Interface cycle to fetch. Memory access instructions require additional cycles to transfer data. Because data transfer cycles cannot overlap, these additional cycles delay instruction fetch and data transfer for other instructions. As a result, the effective execution time of memory access instructions is the number of Memory Interface cycles required. For example, Load instructions require two Memory Interface cycles, and so have an effective execution time of two cycles.

Figure 2.3
L64801 Block Diagram



The four pipeline stages are designated Fetch, Decode, Execute, and Write. The operations performed in these four stages are described below. The Decode, Execute, and Write stages each have an instruction register and an instruction decoder. Each instruction decoder generates signals that control operations later during its own stage and early in the next stage.

Fetch Stage

The Memory Interface fetches an instruction from memory into a CPU register. If the Decode stage is not busy, the instruction is loaded directly into the Decode Instruction Register (DIR). If the Decode stage is busy with the execution of a multiple-cycle instruction, the fetched instruction is queued in the two-word Instruction Buffer (IB1 and IB2). When the

Decode stage becomes available, each of the queued instructions is loaded from the Instruction Buffer into the Decode Instruction Register.

Decode Stage

During the Decode stage, the processor performs three types of operations in parallel: decoding the current instruction, getting the operands required by the current instruction, and calculating instruction addresses for subsequent instructions.

Decoding a single-cycle instruction is straightforward. Decoding a multiple-cycle instruction may generate one or more Internal Operations (IOPs). A multiple-cycle instruction may also generate internal no operations known as Null Cycles. IOPs and Null Cycles are loaded into the Decode Instruction Register and flow through the Decode, Execute, and Write pipeline stages like single-cycle instructions.

In parallel with instruction decode, the processor reads the instruction's operands from the Register File. Operand register A is loaded with Source Register R1 from the Register File. Operand register B is either loaded with Source Register R2 from the Register File or loaded with immediate data from the instruction. When an instruction uses the result of another instruction that has not yet exited the pipeline, the bypass paths load the result into either operand register A or operand register B directly from the appropriate stage. The bypass paths are used when either Source Register R1 or Source Register R2 of the instruction in the Decode stage is the Destination Register of an instruction in the Execute or Write stages.

At the same time, the Address Generator calculates the next instruction address. During the execution of most instructions, the Address Generator increments the most recent instruction address by four and loads the resulting instruction address into the Address Register. Branches, Calls, and memory access instructions are all handled differently from other instructions.

During the execution of Branches and Calls, the Address Generator adds the contents of the Program Counter to the displacement field of the instruction and loads the resulting instruction address into the Address Register.

At the beginning of memory access instructions, such as Load or Store instructions, the Address Generator increments the most recent instruction

address by four and saves the resulting instruction address in the PC Buffer. When the Address Register becomes available at the end of the memory access instruction, the Address Generator loads the PC Buffer into the Address Register.

Execute Stage

The operations performed by the processor during the Execute stage depend on the type of instruction that is being executed. During execution of Arithmetic and Logical instructions, either the Arithmetic/Logic Unit (ALU) or the Shifter processes the contents of operand registers A and B, and stores the result into the Result Register. During execution of Jump and Link, Return from Trap, and memory access instructions, the Address Generator adds together the contents of operand registers A and B and loads the resulting address into the Address Register. During execution of Store Byte and Store Halfword instructions, the Shifter shifts the partial word in operand register A to the correct position within a word and loads it into the Result Register.

Write Stage

During the Write stage, the IU passes the contents of the Result Register through the Load Data Aligner, and writes them to the Destination Register in the Register File. When instruction execution results in a trap or exception, IU does not change the Destination Register. During the Write stage of Load Byte, Load Halfword, and Atomic Load-Store instructions, the Load Data Aligner aligns the load data. As the partial word in the Result Register passes through the Load Data Aligner, the partial word is right-justified and extended to the left with sign or zero bits.

2.4 Instruction Set

The L64801 IU is the primary data processing engine in a SPARC system. The IU executes five types of SPARC instructions: load/store, arithmetic/logical/shift, control-transfer, read/write control register, and floating-point/coprocessor. These instruction types are summarized below.

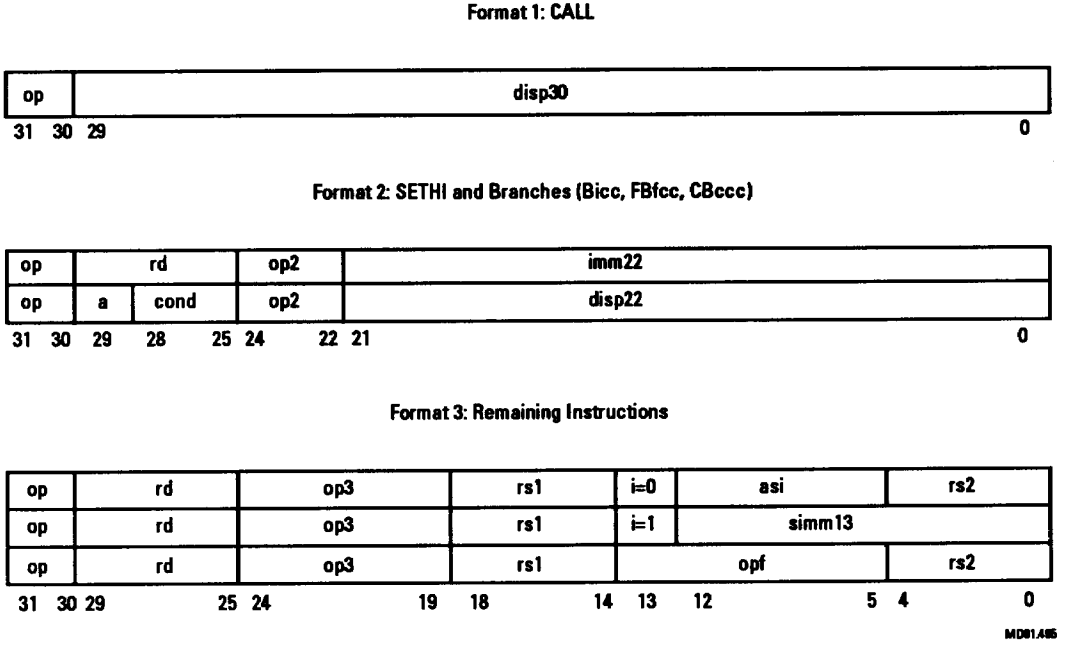
1. Load/Store Instructions: These instructions access memory. The load/store instructions add either the contents of two registers or the contents of one register with a 13-bit signed immediate value to generate the memory address. The instruction destination register field specifies the destination of a load or source for a store. Integer load and store

instructions support 8-, 16-, 32-, and 64-bit transfers. Floating-point instructions support 32- and 64-bit transfers.

2. Arithmetic/Logical/Shift Instructions: These instructions perform arithmetic, tagged arithmetic, logical, and shift operations. They compute a result that is a function of two source operands and either write the result into a destination register or discard it. The tagged arithmetic instructions are useful in artificial intelligence applications. The shift instructions shift the contents of any register left or right any number of bits in one clock cycle, as specified by a register or by the 13-bit signed immediate value.
3. Control-Transfer Instructions: Control-transfer instructions include branches, calls, jumps, and traps. Control transfer is usually delayed so that the instruction immediately following the control-transfer (called the delay instruction) is executed before control is transferred to the target location. Branch and call instructions use program counter relative displacements. The branch instruction provides a displacement of plus or minus eight megabytes. The call instruction's 30-bit displacement allows transfer to any address. The Jump and Link instruction uses a register indirect displacement, as does the Return from Trap instruction. The target address is either the sum of the contents of two registers or the sum of the contents of a register with a 13-bit signed immediate value.
4. Read/Write Control/Status Register Instructions: These instructions access the various control/status registers within the L64801. These registers include the Multiply Step Register, Processor State Register, Trap Base Register, and Window Invalid Mask Register.
5. Floating-point/Coprocessor Instructions: These instructions include floating-point calculations, operations on floating-point registers, and instructions involving the optional coprocessor. Floating-point operations execute concurrently with IU instructions and with other floating-point operations when necessary.

All instructions are 32 bits wide and aligned on 32-bit boundaries in memory. Instructions are classified into three major formats, two of which include sub-formats. Each instruction format is shown in Figure 2.4 with its fields and bit positions. For a complete description of the use of the various instructions and instruction fields, see the *SPARC Architecture Manual.*

Figure 2.4
Instruction Formats



The fields within the instruction formats are defined below:

a **29**

This bit is the annul bit in Format 2 instructions. This bit affects the operation of the instruction encountered immediately after a control transfer.

asi **[12:5]**

This eight-bit field is the address space identifier. Load/store alternate instructions generate this field.

cond **[28:25]**

This field selects the condition code for Format 2 instructions.

disp22 **[21:0]**
disp30 **[29:0]**

These fields are 22-bit and 30-bit sign-extended word displacements for branches and PC-relative calls, respectively.

i **13**

This bit selects the second ALU operand for non-FPop instructions. If i = 0, the second operand is r[rs2]. If i = 1, the second operand is sign-extended simm13.

imm22 **[21:0]**

This field is a 22-bit constant value used by the SETHI instruction.

op **[31:30]**

This field places the instruction into one of the three major formats as shown in the following table.

op	*Format*	*Instruction*
00	2	Bicc, FBfcc, CBccc, SETHI
01	1	CALL
1x	3	Remaining instructions

opf **[13:5]**

This nine-bit field identifies a floating-point operate (FPop) instruction or a coprocessor operate (CPop) instruction.

op2 **[24:22]**

This field comprises bits 24 through 22 of Format 2 instructions. The instructions are encoded as follows:

op2	*Instruction*
000	UNIMP
010	Bicc
100	SETHI
110	FBfcc
111	CBccc

op3 **[24:19]**

This field selects one of the Format 3 opcodes.

rd **[29:25]**

For store instructions, this field selects an R register (or an R register pair) or an F register (or an F register pair) as the source. For all other instructions, this field selects an R register (or R register pair) or an F register (or F register pair) as the destination.

rs1 **[18:14]**

This five-bit field selects the first source operand from either the R registers or the F registers.

rs2 **[4:0]**

This five-bit field selects the second source operand from either the R registers or the F registers.

simm13 **[12:0]**

This field is a sign-extended, 13-bit immediate value used as the second ALU operand when i = 1.

Table 2.2 summarizes the instruction set for the L64801 IU. Refer to the section entitled "Floating-Point Instructions" in Chapter 3, "L64804 Floating-Point Unit," for a listing of the FPU instruction set. For a complete description of the instruction set, refer to the *SPARC Architecture Manual.*

Table 2.2
L64801 Instruction Set

Instruction		*Mnemonic*	*Format*	*op*	*op2*	*op3*	*Cycles*
Add		ADD	3	2		00	1
Add and Modify Integer Condition Code		ADDcc	3	2		10	1
Add with Carry		ADDX	3	2		08	1
Add with Carry and Modify Integer Condition Code		ADDXcc	3	2		18	1
And		AND	3	2		01	1
And and Modify Integer Condition Code		ANDcc	3	2		11	1
And Not		ANDN	3	2		05	1
And Not and Modify Integer Condition Code		ANDNcc	3	2		15	1
Atomic Load-Store Unsigned Byte		LDSTUB	3	3		0D	4
Atomic Load-Store Unsigned Byte into Alternate Space[1]		LDSTUBA	3	3		1D	4
Branch on Coprocessor Condition[2]		CBxxx	2	0	7		–
Branch on Integer Condition	(taken)	Bxxx	2	0	2		1[3]
	(not taken)						2[4]
Call		CALL	1	1			1
Coprocessor Operate[2]		CPop	Cp	2		37	–
Exclusive Nor		XNOR	3	2		07	1

1. Privileged Instruction
2. Coprocessor Instructions cause a Coprocessor_Disabled trap and are implemented in the trap handler
3. Branch Always increases by one cycle when the annul bit a = 1
4. Branch not taken increases by one cycle when the annul bit a = 1

Continued on next page.

Table 2.2
L64801 Instruction
Set, continued

Instruction	*Mnemonic*	*Format*	*op*	*op2*	*op3*	*Cycles*
Exclusive Nor and Modify Integer Condition Code	XNORcc	3	2		17	1
Exclusive Or	XOR	3	2		03	1
Exclusive Or and Modify Integer Condition Code	XORcc	3	2		13	1
Instruction Cache Flush	IFLUSH	3	2		3B	
Inclusive Or	OR	3	2		02	1
Inclusive Or and Modify Integer Condition Code	ORcc	3	2		12	1
Inclusive Or Not	ORN	3	2		06	1
Inclusive Or Not and Modify Integer Condition Code	ORNcc	3	2		16	1
Jump and Link	JMPL	3	2		38	2
Load Coprocessor Register[1]	LDC	3	3		30	–
Load Coprocessor State Register[1]	LDCSR	3	3		31	–
Load Double Coprocessor register[1]	LDDC	3	3		33	–
Load Doubleword	LDD	3	3		03	3^2
Load Doubleword from Alternate Space[3]	LDDA	3	3		13	3^2
Load Signed Byte	LDSB	3	3		09	2^4
Load Signed Byte from Alternate Space[3]	LDSBA	3	3		19	2^4
Load Signed Halfword	LDSH	3	3		0A	2^4
Load Signed Halfword from Alternate Space[3]	LDSHA	3	3		1A	2^4
Load Unsigned Byte	LDUB	3	3		01	2^4
Load Unsigned Byte from Alternate Space[3]	LDUBA	3	3		11	2^4
Load Unsigned Halfword	LDUH	3	3		02	2^4
Load Unsigned Halfword from Alternate Space[3]	LDUHA	3	3		12	2^4
Load Word	LD	3	3		00	2^4
Load Word from Alternate Space[3]	LDA	3	3		10	2^4
Multiply Step and Modify Integer Condition Code	MULScc	3	2		24	1
Read Processor State Register[3]	RDPSR	3	2		29	1
Read Trap Base Register[3]	RDTBR	3	2		2B	1
Read Window Invalid Mask register[3]	RDWIM	3	2		2A	1
Read Y Register	RDY	3	2		28	1
Restore Caller's Window	RESTORE	3	2		3D	1
Return from Trap[3]	RETT	3	2		39	2
Save Caller's Window	SAVE	3	2		3C	1

1. Coprocessor Instructions cause a Coprocessor_Disabled trap and are implemented in the trap handler
2. Load Doubleword increases by one cycle when the second word of load data is used by the instruction immediately following
3. Privileged Instruction
4. Load (except Doubleword) increases by one cycle when the load data are used by the instruction immediately following

Continued on next page.

Table 2.2
L64801 Instruction
Set, continued

Instruction	*Mnemonic*	*Format*	*op*	*op2*	*op3*	*Cycles*
Set High	SETHI	2	0	4		1
Shift Left Logical	SLL	3	2		25	1
Shift Right Arithmetic	SRA	3	2		27	1
Shift Right Logical	SRL	3	2		26	1
Store Byte	STB	3	3		05	3
Store Byte into Alternate Space[1]	STBA	3	3		15	3
Store Coprocessor[1]	STC	3	3		34	–
Store Coprocessor State Register[2]	STCSR	3	3		35	–
Store Double Coprocessor[2]	STDC	3	3		37	–
Store Double Coprocessor Queue[1,2]	STDCQ	3	3		36	–
Store Doubleword	STD	3	3		07	4
Store Doubleword into Alternate Space[1]	STDA	3	3		17	4
Store Halfword	STH	3	3		06	3
Store Halfword into Alternate Space[1]	STHA	3	3		16	3
Store Word	ST	3	3		04	3
Store Word into Alternate Space[1]	STA	3	3		14	3
Subtract	SUB	3	2		04	1
Subtract and Modify Integer Condition Code	SUBcc	3	2		14	1
Subtract with Carry	SUBX	3	2		0C	1
Subtract with Carry and Modify Integer Condition Code	SUBXcc	3	2		1C	1
Swap R Register with Memory	SWAP	3	3		0F	
Swap R Register with Alternate Space Memory[1]	SWAPA	3	3		1F	
Tagged Add and Modify Integer Condition Code	TADDcc	3	2		20	1
Tagged Add, Modify Integer Condition Code, and Trap on Overflow	TADDccTV	3	2		22	1
Tagged Subtract and Modify Integer Condition Code	TSUBcc	3	2		21	1
Tagged Subtract, Modify Integer Condition Code, and Trap on Overflow	TSUBccTV	3	2		23	1
Trap on Integer Condition (taken)	Txxx	3	2		3A	4
(not taken)						1
Unimplemented	UNIMP	2	0	0		
Write Processor State Register[1]	WRPSR	3	2		31	1
Write Trap Base Register[1]	WRTBR	3	2		33	1
Write Window Invalid Mask register[1]	WRWIM	3	2		32	1
Write Y Register	WRY	3	2		30	1

1. Privileged Instruction
2. Coprocessor Instructions cause a Coprocessor_Disabled trap and are implemented in the trap handler

2.5 Trap and Exception Handling

The L64801 generates traps in response to both internal and external events. These traps switch control from the instruction stream to an address in a trap table. The only exception is the reset trap, which transfers control to address zero.

Traps fall into three categories: synchronous, asynchronous, and floating-point. Either hardware or Trap on Integer Condition Code (Ticc) instructions cause synchronous traps. Synchronous traps occur during the instruction that caused them. Interrupt requests on the $\overline{IRL}$[3:0] inputs cause asynchronous traps. The L64801 synchronizes asynchronous traps and services them after the current instruction has completed. FPop instructions cause floating-point traps. Floating-point traps occur before the instruction is completed. However, since the IU and FPU are operating concurrently, other non-floating-point instructions may have executed in the meantime.

Synchronous Traps

The IU generates synchronous traps in response to internal conditions, external signals, or Trap (Ticc) instructions. The IU takes these traps immediately. When an instruction (other than Ticc) causes a trap, the L64801 aborts the instruction before the state of the processor is changed.

If a synchronous trap occurs while traps are disabled (PSR[ET] = 0), the IU enters the error state. The L64801 saves the contents of its internal registers as if a trap had occurred; then it stops and drives $\overline{ERROR}$ LOW. The only way to exit the error state is to force a Reset trap by driving $\overline{RESET}$ LOW. The Reset trap transfers control to address zero, but does not change the information that was loaded into the processor's internal registers when it entered the error state. This information determines the cause of the error.

External hardware such as the memory subsystem may cause synchronous traps. Table 2.3 lists the synchronous traps that occur in response to external signals:

Table 2.3 Synchronous Traps and Exceptions

Trap	*Initiating Signal*	*Condition*
Data_Access_Exception	$\overline{MEXC}$	Error during data access
Instruction_Access_Exception	$\overline{MEXC}$	Error during instruction access

Asynchronous Traps

The L64801 generates asynchronous traps in response to the Interrupt Request (IRL[3:0]) inputs. The IU waits for the currently executing instruction to complete before processing the trap. The IRL[3:0] inputs provide 15 levels of interrupts.

Setting IRL[3:0] all HIGH signifies interrupt level 0 (no interrupt). Setting IRL[3:0] all LOW signifies interrupt level 15, which is a non-maskable interrupt. All other combinations represent interrupt requests that can be masked by the PIL field in the PSR. For the interrupt to be taken, the Interrupt Request Level, which is the complement of IRL[3:0], must be greater than the value in the PIL field of the PSR.

Floating-Point Traps

The IU generates floating-point traps in response to the $\overline{\text{FEXC}}$ input. The FPU asserts $\overline{\text{FEXC}}$ upon recognizing a floating-point exception. The IU asserts FXACK to acknowledge the trap. At this time, the FPU enters an exception mode state. To exit this state, execute one or more STDFQ instructions in order to empty the Floating-Point Queue.

The PC that corresponds to a floating-point exception always points to a floating-point instruction. However, the exception itself is always due to a previously executed floating-point instruction. The instruction and the value of the PC from which the instruction was fetched are in the Floating-Point Queue.

Since the IU and FPU operate concurrently, the floating-point exception typically occurs sometime after the exception-causing instruction. The IU does not take the exception trap until another floating-point instruction is encountered.

Trap Addressing

Each type of trap is assigned a priority. When multiple trap requests occur, the IU takes the highest priority trap and ignores lower priority traps. To ensure recognition by the L64801, lower priority traps must either persist or be repeated.

When the L64801 recognizes a trap, it performs the following actions:

1. It disables traps (PSR[ET] $\leftarrow$ 0).
2. It copies the Supervisor (S) bit of the PSR into the Previous Supervisor (PS) bit, and then sets the S bit to 1 (Supervisor mode).

3. It decrements the Current Window Pointer (CWP) field of the PSR by 1 modulo 7. CWP then points to a new window.
4. It saves the PC into R[17] and the nPC into R[18] of the new window.
5. It sets the Trap Type (TT) field of the Trap Base Register (TBR) to the appropriate value. The Trap Type values are shown in Traps and Priorities (Table 2.4) below.
6. If the trap is not a reset, it loads the PC with the contents of the TBR, and the nPC with the contents of the TBR plus four. If the trap is a reset, it loads the PC with zero and the nPC with four.

Table 2.4 shows the traps and their priorities. For most traps, the IU writes the trap number into the TT field of the TBR. For a Ticc instruction, the instruction calculates the TT value.

Table 2.4
Traps and Priorities

Trap	*Priority*	*Trap Type (TT)*	*Synchronous or Asynchronous*
Reset	1	-	Async
Instruction_Access_Exception	2	1	Sync
Illegal_Instruction	3	2	"
Privileged_Instruction	4	3	"
Floating-Point_Disabled	5	4	"
Coprocessor_Disabled	5	36	"
Window_Overflow	6	5	"
Window_Underflow	7	6	"
Memory_Address_Not_Aligned	8	7	"
Floating-Point_Exception	9	8	"
Coprocessor_Exception[1]	9	40	"
Data_Access_Exception	10	9	"
Tag_Overflow	11	10	"
Trap_Instruction (Ticc)	12	128-255	"
Interrupt_Level_15	13	31	Async
Interrupt_Level_14	14	30	"
Interrupt_Level_13	15	29	"
Interrupt_Level_12	16	28	"
Interrupt_Level_11	17	27	"
Interrupt_Level_10	18	26	"
Interrupt_Level_9	19	25	"
Interrupt_Level_8	20	24	"
Interrupt_Level_7	21	23	"
Interrupt_Level_6	22	22	"
Interrupt_Level_5	23	21	"
Interrupt_Level_4	24	20	"
Interrupt_Level_3	25	19	"
Interrupt_Level_2	26	18	"
Interrupt_Level_1	27	17	"

1. The L64801 implements these traps in software

2.6 External Signals

This section describes the signals of the L64801. These signals are described in alphabetical order. Signals that are active LOW are marked with an overbar; all others are active HIGH. For example $\overline{\text{NAME}}$ is active LOW and NAME is active HIGH.

A[31:0] **IU Address [31:0]** **Output**

A[31:0] comprise the address portion of the Local Bus. The IU uses these signals to specify the addresses of instructions or data. During an instruction fetch cycle, the bus contains an instruction address, and during a load or store data cycle, A[31:0] contains a data address. $\overline{\text{AOE}}$ controls the output drivers for A[31:0]. The address bus remains valid during all data cycles of loads, stores, load doubles, and atomic load/stores. In systems with cache, the low bits of the address read the cache RAMs and cache tags, and the high bits of the address compare the tags.

$\overline{\text{AOE}}$ **Address Output Enable** **Input**

$\overline{\text{AOE}}$ controls the output drivers for A[31:0] and ASI[7:0]. External logic drives $\overline{\text{AOE}}$ LOW during normal operation and drives $\overline{\text{AOE}}$ HIGH when the bus is granted to another bus master (that is, when either $\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, or $\overline{\text{BHOLD}}$ is asserted).

ASI[7:0] **Address Space Identifier [7:0]** **3-State Output**

The IU uses the eight ASI bits to specify the address space for an instruction fetch or data access to memory. During any given cycle, ASI[7:0] specify the address space that corresponds to the address on A[31:0] during that same cycle. The IU does not latch these signals, which the IU outputs at the rising edge of CLK.

ASI[7:0] are three-stated when $\overline{\text{ASIOE}}$ is HIGH.

Of the 256 possible address spaces that these signals represent, the IU automatically issues four address spaces during instruction fetches and standard memory accesses. The table below lists the ASIs.

ASI[7:0]	*Address Space*
0x00 – 0x07	Implementation Definable
0x08	User Instruction
0x09	Supervisor Instruction
0x0A	User Data
0x0B	Supervisor Data
0x0C – 0xFF	Implementation Definable

During the data cycles of alternate load and store instructions, ASI[7:0] contain the space identifier specified by the instruction opcode.

$\overline{\text{ASIOE}}$ **Address Space Identifier Output Enable** **Input**
$\overline{\text{ASIOE}}$ controls the output drivers for ASI[7:0]. External logic drives $\overline{\text{AOE}}$ LOW during normal operation and drives $\overline{\text{AOE}}$ HIGH when the bus is granted to another bus master (that is, when either $\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, or $\overline{\text{BHOLD}}$ is asserted).

$\overline{\text{BHOLD}}$ **Bus Hold** **Input**
External logic (for example a bus controller) asserts $\overline{\text{BHOLD}}$ to the Local Bus Master. When asserted, $\overline{\text{BHOLD}}$ freezes the processor pipeline. When $\overline{\text{BHOLD}}$ is deasserted, external logic must guarantee that all inputs to the IU are in the same state as they were before the assertion of $\overline{\text{BHOLD}}$. External logic must assert $\overline{\text{BHOLD}}$ after the rising edge of CLK (the start of the clock cycle) and keep the signal asserted until after the falling edge of CLK (the middle of the cycle). The IU latches $\overline{\text{BHOLD}}$ before using it.

CLK **Clock** **Input**
This 50-percent duty-cycle clock synchronizes all system operations, including the IU, MCT, FPU, and transactions on the Mbus. It is HIGH during the first half of the processor cycle, and LOW during the second half. The rising edge of CLK defines the beginning of each pipeline stage in the IU chip.

D[31:0] **IU Data Bus [31:0]** **3-State Bidirectional**
D[31:0] is a bidirectional bus that comprises the data portion of the Local Bus. D31 corresponds to the most significant bit of the 32-bit word.

The IU drives the bus when executing integer store instructions and during the store cycle of atomic load/store instructions. D[31:0] are valid at the rising edge of CLK. Once latched by external logic, store data are valid during the second data cycle of a store word access, the second and third data cycle of a store doubleword access, and the third data cycle of an atomic load/store access.

The IU samples the bus when executing integer load instructions, when fetching instructions, and during the load cycles of atomic load/store instructions.

The IU aligns the data internally for both load and store instructions. It aligns a doubleword on an eight-byte boundary, a word on a four-byte boundary, and a halfword on a two-byte boundary. If a

doubleword, word, or halfword load or store instruction generates an improperly aligned address, the IU generates a Memory Address Not Aligned Trap. Instructions and operands are always fetched from modulo-32 addresses.

DFETCH **Data Fetch** **Output**
The IU asserts DFETCH to indicate that the current bus cycle is a data transfer cycle. The L64801 deasserts DFETCH to indicates that the current bus cycle is an instruction cycle. The IU asserts NULL_CYC to nullify an instruction or data cycle.

$\overline{\text{DOE}}$ **Data Output Enable** **Input**
External logic deasserts $\overline{\text{DOE}}$ to three-state the IU output drivers for D[31:0]. External logic should deassert this signal only when the bus is granted to another bus master (that is, when either $\overline{\text{MHOLDA}}$, $\overline{\text{MHOLDB}}$, $\overline{\text{MHOLDC}}$, $\overline{\text{SHOLD}}$, or $\overline{\text{BHOLD}}$ is asserted). $\overline{\text{DOE}}$ must be asserted during normal operations.

$\overline{\text{ERROR}}$ **Error** **Output**
If traps are disabled (the ET bit in the PSR is 1), the IU asserts this signal when a trap is encountered. In this situation, the IU saves the PC and nPC, sets the TT value in the TBR, enters into an error state, asserts $\overline{\text{ERROR}}$ and halts. Assert $\overline{\text{RESET}}$ to restart the IU.

F[31:0] **Floating-Point Bus** **Output**
This 32-bit bus sends floating-point instructions and addresses to the L64804. Each floating-point instruction uses this bus for two cycles: the first cycle carries the instruction and the second cycle carries the address.

FADR **Floating-Point Address** **Output**
The IU asserts FADR during the cycle that F[31:0] contains a valid floating-point instruction address. When FADR is asserted, the L64804 latches the address into its address register.

FCC[1:0] **Floating-Point Condition Codes** **Input**
The FCC[1:0] bits contain the current condition code of the L64804 FPU. These bits are valid only when FCCV is HIGH.

The IU checks FCC[1:0] when executing an FBfcc instruction. If FCC[1:0] are not valid during the execute cycle, the IU delays the execute cycle. The following table lists the condition code encoding.

FCC1	*FCC0*	*Condition*
0	0	Equal
0	1	op1 < op2
1	0	op1 > op2
1	1	Unordered

FCCV **Floating-Point Condition Code Valid** **Input**
The L64804 FPU asserts the FCCV signal when FCC[1:0] contain a valid condition. The FPU deasserts FCCV if pending floating-point compare instructions are in the Floating-Point Queue. The L64804 reasserts FCCV when the compare instruction completes and FCC[1:0] contain valid data.

FEND **End Floating-Point Instruction** **Output**
The IU asserts FEND during the last cycle of a floating-point instruction in the IU pipeline. The FPU uses FEND to synchronize the instruction/address in the FPU pipeline with the IU pipeline.

$\overline{\text{FEXC}}$ **Floating-Point Exception** **Input**
The FPU asserts $\overline{\text{FEXC}}$ when a floating-point exception occurs. $\overline{\text{FEXC}}$ remains asserted until the IU asserts FXACK, which indicates the IU has taken a trap. Floating-point exceptions are taken only during the execution of floating-point instructions.

$\overline{\text{FHOLD}}$ **Floating-Point Hold** **Input**
The FPU asserts $\overline{\text{FHOLD}}$ if it cannot continue execution due to a resource or operand dependency. The FPU checks for all dependencies in the write stage and, if necessary, asserts $\overline{\text{FHOLD}}$ in the same cycle. When $\overline{\text{FHOLD}}$ is asserted, the IU freezes its pipeline in the next cycle. The IU releases its pipeline when the FPU deasserts $\overline{\text{FHOLD}}$.

FINS **Floating-Point Instruction** **Output**
When the IU encounters a floating-point instruction, it asserts FINS during the IU's Execute stage. At this time, F[31:0] contains a valid floating-point instruction. When FINS is asserted, the FPU latches the instruction from F[31:0] into its instruction register.

FLUSH **Floating-Point Instruction Flush** **Output**
The IU asserts FLUSH to tell the FPU to flush the instructions in the FPU instruction registers. This operation may happen when the IU takes a trap. The IU restarts the flushed instructions after returning from the trap. The FLUSH signal does not affect instructions in the floating-point queue.

$\overline{FP}$ **Floating-Point Present** **Input w/Pullup**
$\overline{FP}$ indicates whether an FPU is present in the system (0 = FPU; 1 = no FPU). $\overline{FP}$ is tied to VDD through an internal resistor. The IU generates an fp_disabled trap if $\overline{FP}$ is HIGH during the execution of a floating-point instruction.

FXACK **Floating-Point Exception Acknowledge** **Output**
The IU asserts FXACK to acknowledge to the FPU that the current $\overline{FEXC}$ trap is taken.

$\overline{HAL}$ **Hold Address Latch** **Output**
The IU asserts $\overline{HAL}$ in order to freeze the clock to the external memory address register. $\overline{HAL}$ is asserted during the execution of some multiple-cycle instructions, internal interlocks, and whenever at least one of the hold signals ($\overline{MHOLDA}$, $\overline{MHOLDB}$, $\overline{MHOLDC}$, $\overline{SHOLD}$, or $\overline{BHOLD}$) is asserted.

$\overline{IH\_NULL}$ **Null Cycle Reset** **Input**
When asserted, this signal resets NULL_CYC to LOW.

IRL[3:0] **Interrupt Level** **Input**
The data on these pins define the External Interrupt Level. IRL[3:0] equal to 0000_2 indicates that no external interrupts are pending. Because interrupts are asynchronous events, a given interrupt level must remain valid for at least two consecutive cycles for the IU to recognize it. The IU uses two on-chip synchronizing latches to sample these signals on the rising edge of CLK. IRL[3:0] equal to 1111_2 signifies a non-maskable interrupt. All other interrupt levels are maskable. Use the PIL field of the PSR to mask the other levels.

LDST **Load/Store Cycle** **3-State Output**
The IU asserts this signal during all data cycles of atomic load/store instructions. The IU three-states LDST if $\overline{ASIOE}$ is deasserted.

LOCK **Bus Lock Request** **Output**
The IU asserts LOCK during multiple cycle transactions to indicate that no other device should attempt to seize the Local Bus. Multiple cycle transactions include atomic load/stores, doubleword loads, and doubleword stores. Since the IU asserts LOCK to signal an atomic operation, external logic should not assert $\overline{BHOLD}$ until LOCK is deasserted. This signal is valid at the rising edge of CLK.

$\overline{MDS}$ **Memory Data Strobe** **Input**
Memory management hardware asserts $\overline{MDS}$ to indicate that data on D[31:0] are valid during cache misses or slow memory accesses. When asserted, $\overline{MDS}$ enables the clock input to the IU's Instructions

Register (for an instruction fetch) or to the IU's Load Result Register (for a load). Memory management hardware may only assert $\overline{MDS}$ when $\overline{MHOLDA}$, $\overline{MHOLDB}$, $\overline{MHOLDC}$, or $\overline{SHOLD}$ have halted the IU's pipeline.

$\overline{MEXC}$ **Memory Exception** **Input**

The memory or cache controller asserts $\overline{MEXC}$ to indicate to the IU that the memory system was unable to supply a valid instruction or data. The IU latches $\overline{MEXC}$ at the rising edge of CLK and uses $\overline{MEXC}$ in the following cycle. If $\overline{MEXC}$ is asserted during an instruction fetch cycle, the IU generates an instruction access exception. If $\overline{MEXC}$ is asserted during a data fetch cycle, the IU generates a data access exception.

The memory or cache controller may only assert $\overline{MEXC}$ while asserting $\overline{MHOLDA}$, $\overline{MHOLDB}$, or $\overline{MHOLDC}$ and $\overline{MDS}$. When $\overline{MDS}$ is applied with $\overline{MEXC}$, the IU ignores the contents of the data bus. $\overline{MEXC}$ must be deasserted in the clock cycle in which the Hold from Memory signal is released.

$\overline{MHOLDA}$, $\overline{MHOLDB}$, $\overline{MHOLDC}$, $\overline{SHOLD}$ **Hold from Memory** **Input**

Memory management hardware asserts any of these signals in order to freeze the clock to the IU and FPU during a cache miss (for systems with cache) or when a slow memory is accessed. These inputs freeze the IU pipeline and cause the IU outputs to maintain the output values they had at the rising edge of the clock in the cycle before one of the Hold from Memory signals was asserted. The IU logically ORs all hold signals ($\overline{MHOLDA}$, $\overline{MHOLDB}$, $\overline{MHOLDC}$, $\overline{SHOLD}$, $\overline{BHOLD}$) to generate a final $\overline{MHOLD}$ signal for freezing the IU pipeline.

Memory management hardware also uses the $\overline{MDS}$ signal for strobing memory exceptions (see the $\overline{MEXC}$ definition).

NULL_CYC **Null Cycle** **3-State Output**

When asserted, this signal indicates that the IU nullified the current memory address (whose address is held in the external memory address register). NULL_CYC disables cache misses in systems with cache and to handle memory exceptions during the current memory access.

RD **Read Cycle** **3-State Output**

The IU uses this signal to classify the current memory access as a read or write operation (1 = read; 0 = write). The IU deasserts RD only during the data cycles of store instructions and the store cycles of atomic load/store instructions. For atomic load/stores, the RD signal is asserted during the first data cycle (read) and deasserted during the second and third data cycles (write).

In conjunction with SIZE[1:0], ASI[7:0], and LDST, RD can determine the type of bus transaction and check read/write access rights. RD also can turn off the output drivers of data RAMs during a store operation.

RD is three-stated if $\overline{\text{ASIOE}}$ is deasserted.

$\overline{\text{RESET}}$ **Reset** **Input**

The external system asserts RESET in order to reset the IU. The $\overline{\text{RESET}}$ signal must be asserted for a minimum of eight processor clock cycles. On the first rising edge of clock after $\overline{\text{RESET}}$ is driven HIGH, the processor starts fetching instructions from address zero. When $\overline{\text{RESET}}$ is driven LOW, the processor is initialized as follows:

1. Supervisor mode is selected (PSR[S] = 1).
2. The Fetch Program Counter is set to 0.

All other registers and PSR bits are unchanged. At power on, all other registers and PSR bits are undefined until software initializes them.

$\overline{\text{SHOLD}}$ **Hold from Memory** **Input**

See the $\overline{\text{MHOLD}}$ signal description for a definition of the $\overline{\text{SHOLD}}$ signal.

SIZE[1:0] **Size [1:0]** **3-State Output**

SIZE[1:0] represent the data size of the memory address currently on A[31:0]. These bits remain valid during all data cycles of load, store, load-double, store-double, and atomic load/store instructions. Since all instructions are 32 bits long, SIZE[1:0] is set to 10_2 during all instruction fetch cycles. The following table shows the encoding of the SIZE[1:0] bits.

SIZE[1:0] are three-stated if $\overline{\text{ASIOE}}$ is deasserted.

SIZE1	*SIZE0*	*Data Transfer Type*
0	0	Byte
0	1	Halfword
1	0	Word
1	1	Word (Load/Store Double)

$\overline{TC}$ **Trap Condition** **Input**

This signal controls the behavior of the IFLUSH instruction. If $\overline{TC}$ is HIGH, IFLUSH executes like a NOP with no side effects. If $\overline{TC}$ is LOW, IFLUSH causes an unimplemented instruction trap.

$\overline{WE}$ **Write Enable** **3-State Output**

The IU asserts $\overline{WE}$ during the second data cycle of store word instructions, the second and third data cycles of store doubleword instructions, and the third data cycle of atomic load/store instructions. This signal is valid at the rising edge of CLK.

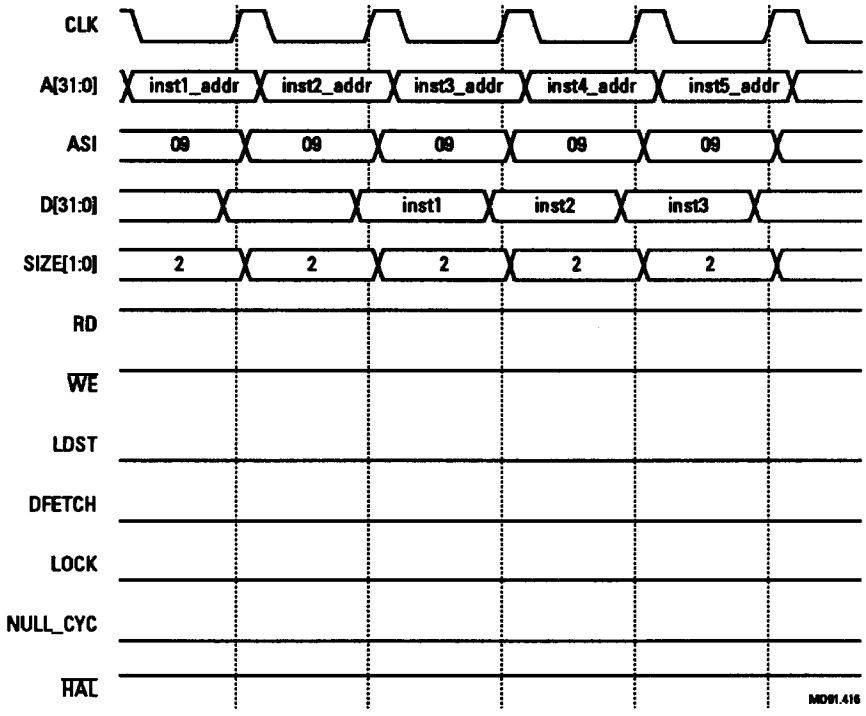
$\overline{WE}$ is three-stated when not asserted and three-stated when $\overline{ASIOE}$ is deasserted.

2.7 Functional Waveforms

This section describes the various memory operations of the L64801 IU. Functional timing diagrams are provided for load operations, store operations, atomic transactions, and floating-point operations. All IU signals are referenced to the rising edge of the clock.

Figure 2.5 shows the functional timing for several instruction fetches. The instruction addresses are presented on the A[31:0] lines. The address space identifier for each instruction is output on the ASI[7:0] lines. The instruction is fetched on the D[31:0] bus. The SIZE[1:0] pins indicate the size of the data transfer.

Figure 2.5 Instruction Fetch Timing



Load Transactions

Figure 2.6 shows the timing for a load single byte instruction. The L64801 outputs the instruction address on A[31:0] during clock T1 and outputs the address space identifier on the following cycle. The IU fetches the LDSB instruction on D[31:0] in T2. The L64801 asserts DFETCH during T4 to indicate that the address refers to the data that is fetched during the next clock cycle, T5, on D[31:0]. SIZE[1:0] equals 0 to indicate the data size is a byte.

Figure 2.6
Load Integer Timing

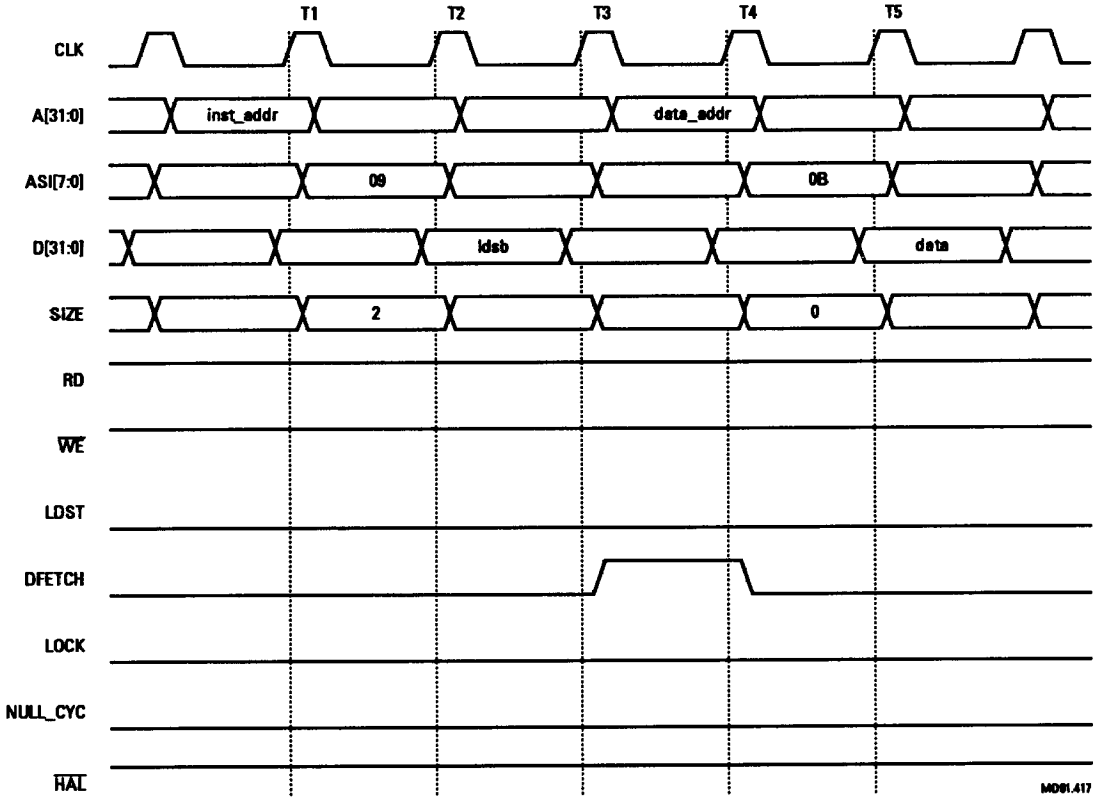


Figure 2.7 shows the timing for a load double integer instruction. This operation is similar to the load integer operation except that the processor uses one additional cycle to repeat the load operation for the second word. The IU fetches the instruction during T2. The IU drives the load address for the first data word (the most significant word) during T4 and drives the load address for the second data word (the least significant word) during T5. The IU asserts DFETCH during T4 to indicate that the address refers to the data that is fetched during T5 and T6. Note that the address of the second data word is equal to the address of the first data word + 4. The IU drives LOCK HIGH to indicate a multiple cycle transaction is occurring. SIZE[1:0] equal 11_2 to indicate a double operand.

Figure 2.7
Load Double Integer
Timing

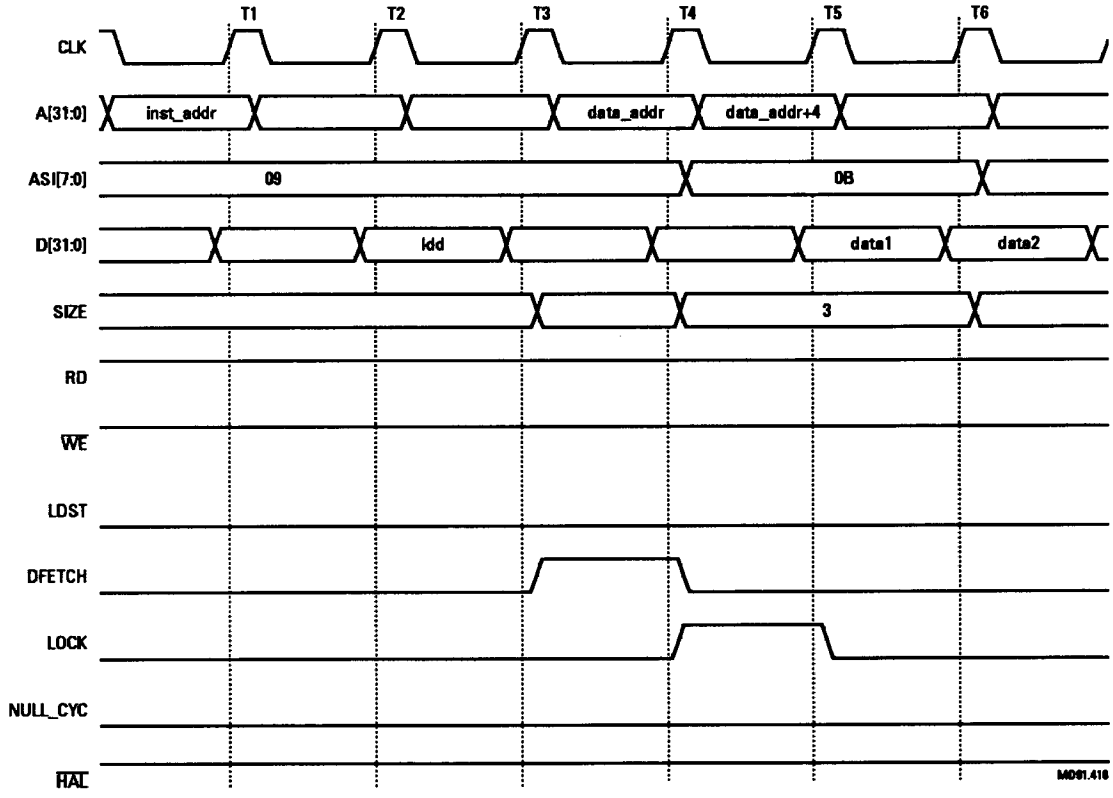


Figure 2.8 shows the timing for a load floating-point instruction. This operation is similar to the load integer instruction except that the IU also generates the floating-point control signals, FINS, FADR, and FEND. The IU asserts FINS during T4 when the floating-point instruction is on F[31:0]. The IU then asserts FADR during T5 to indicate that the floating-point instruction address is on F[31:0].

Figure 2.8
Load Floating-Point Timing

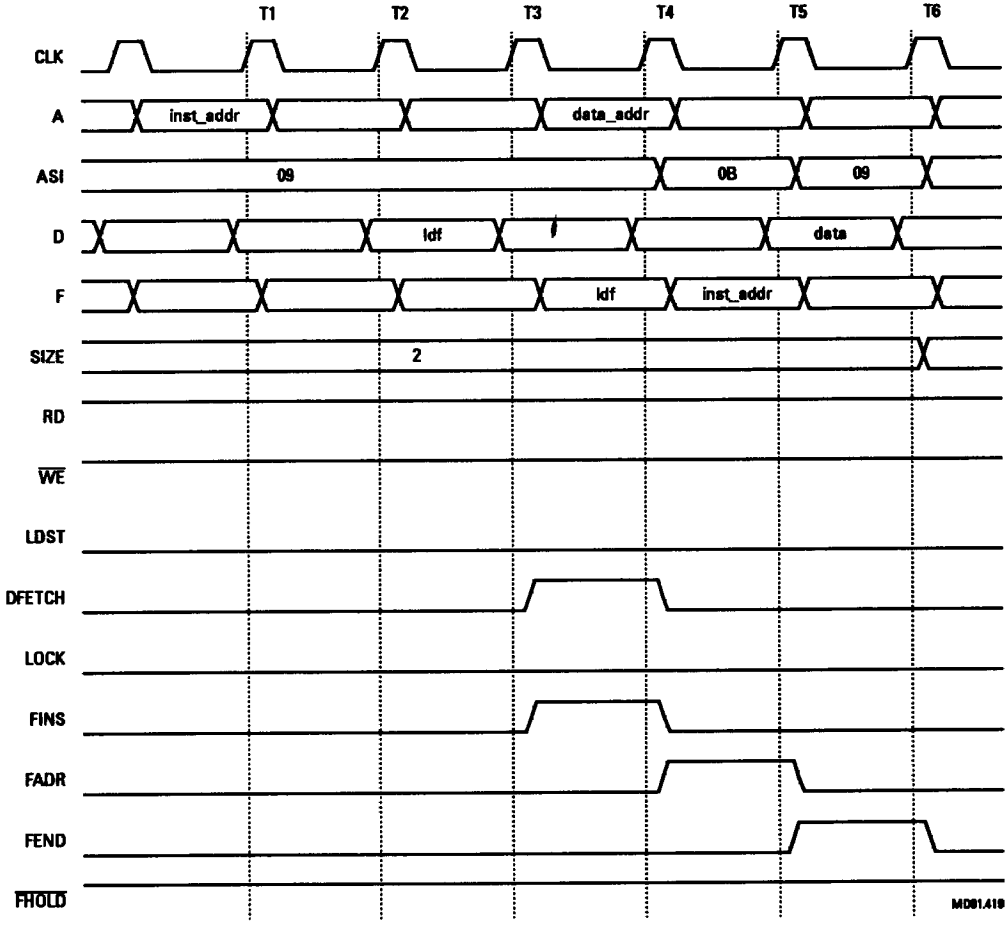
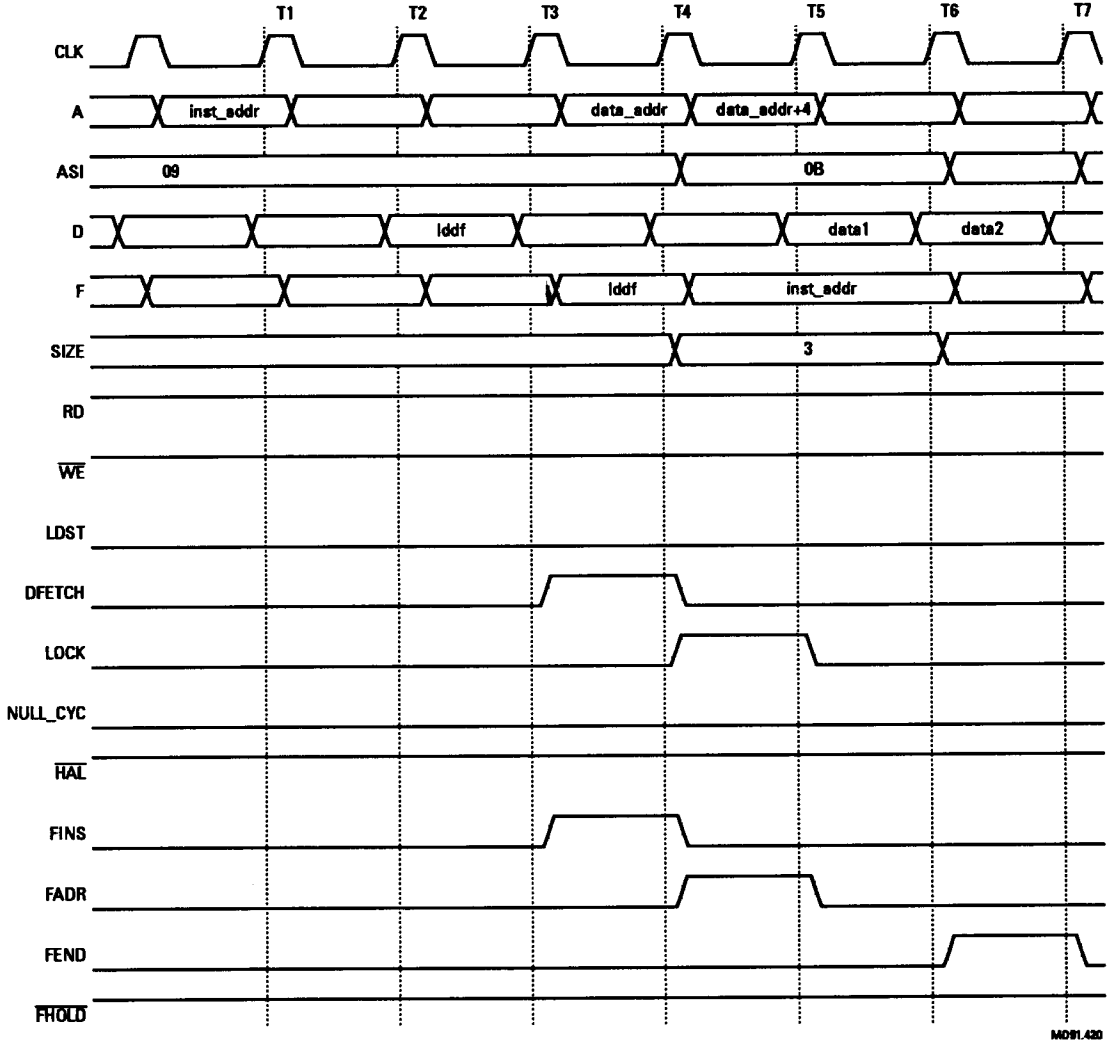


Figure 2.9 shows the timing for a load double floating-point instruction. This operation is similar to the load double integer instruction except that the IU also generates the floating-point signals FINS, FADR, and FEND.

Figure 2.9
Load Double
Floating-Point
Timing



Store Transactions

Figure 2.10 shows a store integer instruction. During T1, the IU places the store address on the A[31:0] bus. During T2, the IU fetches the ST instruction from D[31:0]. During T4, the IU asserts DFETCH to indicate that the data address for the store is on A[31:0]. The IU asserts $\overline{WE}$ and then places the store data on D[31:0]. The data is stored during T6 when the IU deasserts $\overline{WE}$. This operation takes two extra cycles to complete over load integer operations as the processor cannot send both the address and the data simultaneously, and because the processor must wait to see if the store generates an exception or a cache miss.

Figure 2.10
Store Integer Timing

Figure 2.11 shows the timing for a store double integer instruction. This operation is similar to the store integer operation except that the processor uses one additional cycle to repeat the store operation for the second word. The IU fetches the instruction from D[31:0] during T2. The IU drives the store address for the first data word during T4 and drives the store address for the second data word during T5. Note that the address of the second store is equal to the first address + 4. The L64801 drives the data during T6 and T7. D[31:0] contain the most significant word during T6; D[31:0] contain the least significant word during T7. The size bits are set to 11_2 to indicate a double operand.

Figure 2.11
Store Double
Integer Timing

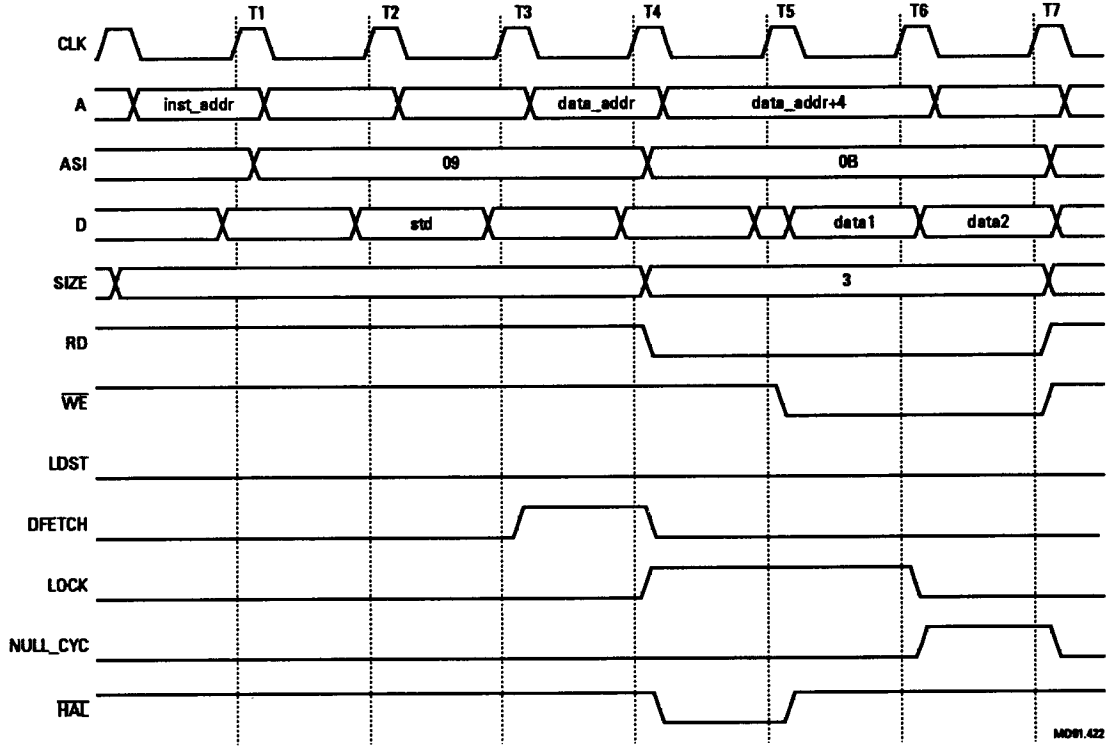


Figure 2.12 shows the timing for a store floating-point operation. This operation is similar to the store integer operation except that the IU also generates the additional floating-point signals, FINS, FADR, and FEND.

Figure 2.12
Store Floating-Point
Timing

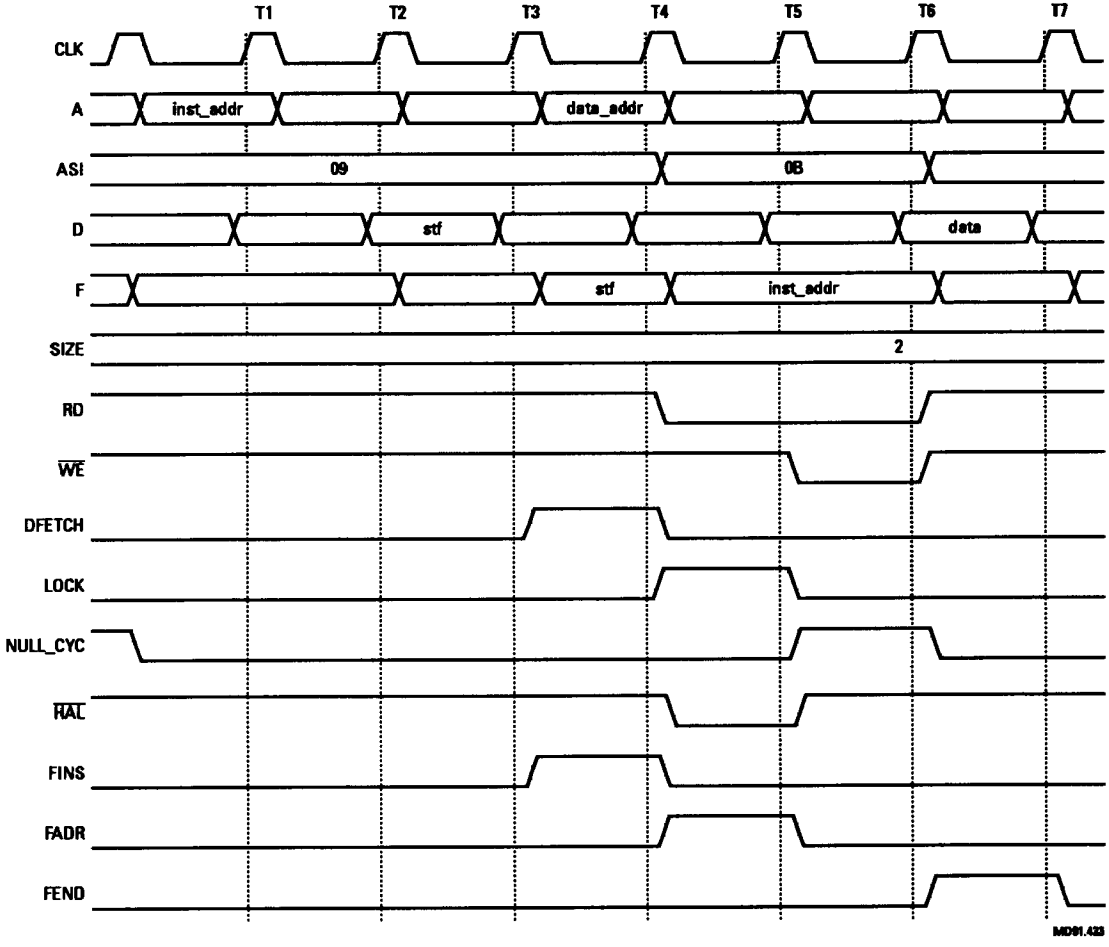


Figure 2.13 shows the timing for a store double floating-point instruction. This operation is similar to the store floating-point operation except that the IU requires an extra cycle to store the second half of the floating-point operand.

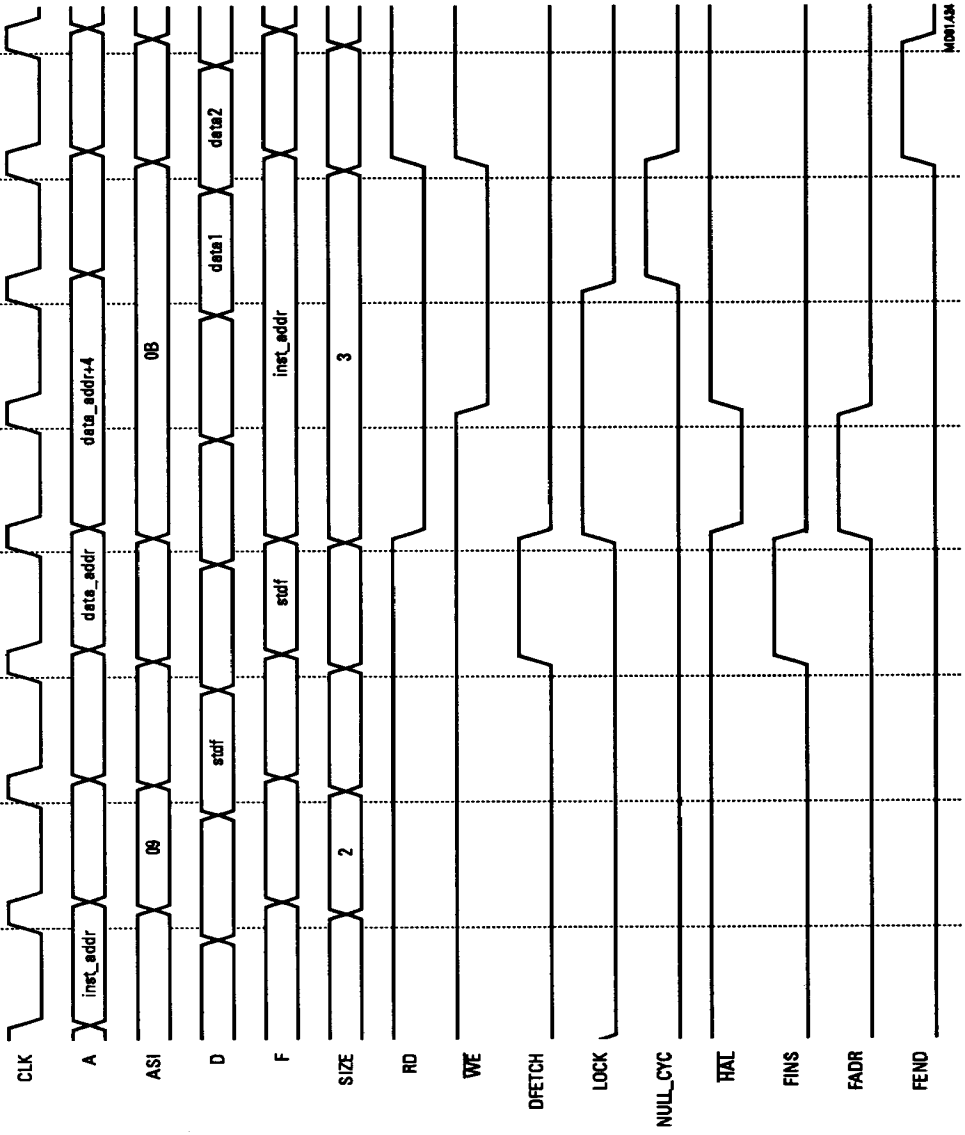
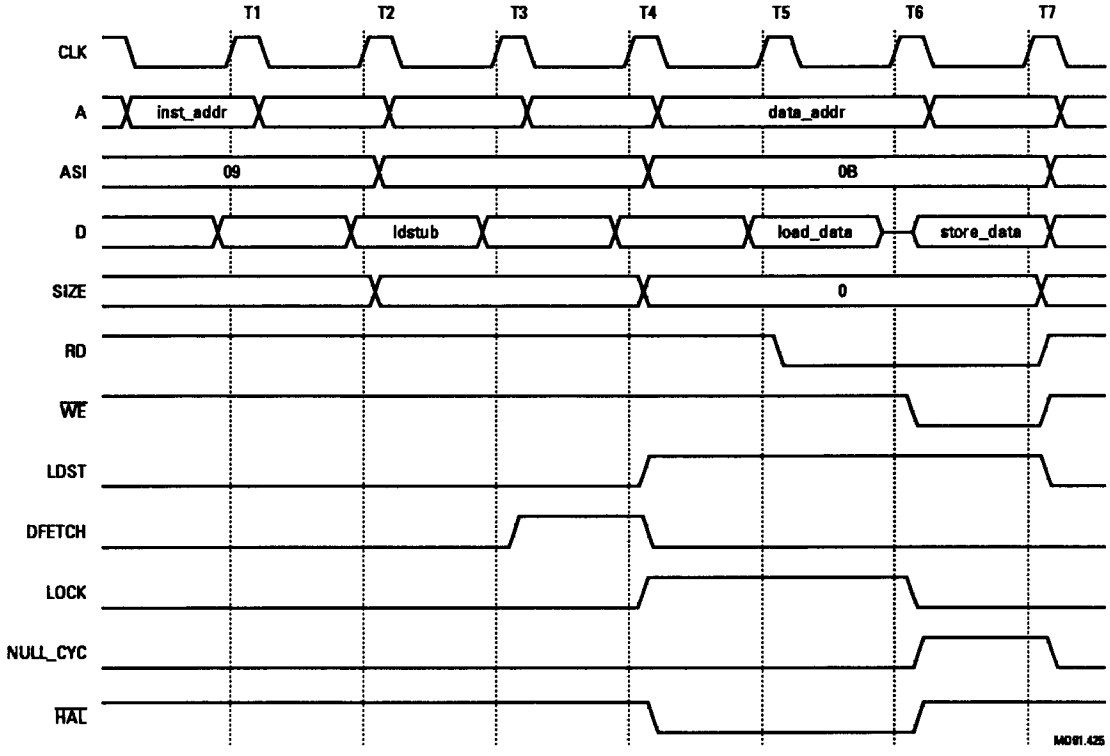


Figure 2.13 Store Double Floating-Point Timing

Atomic Transactions

Atomic transactions consist of two or more uninterrupted steps. Once the sequence has started, it cannot be interrupted. To ensure that the bus is available for the next transaction, the IU asserts LOCK for as long as necessary. The atomic load and store unsigned byte is the only atomic transaction currently supported (see Figure 2.14). This instruction takes seven cycles to complete.

Figure 2.14
Atomic Load/Store Unsigned Byte Timing



Floating-Point Operations

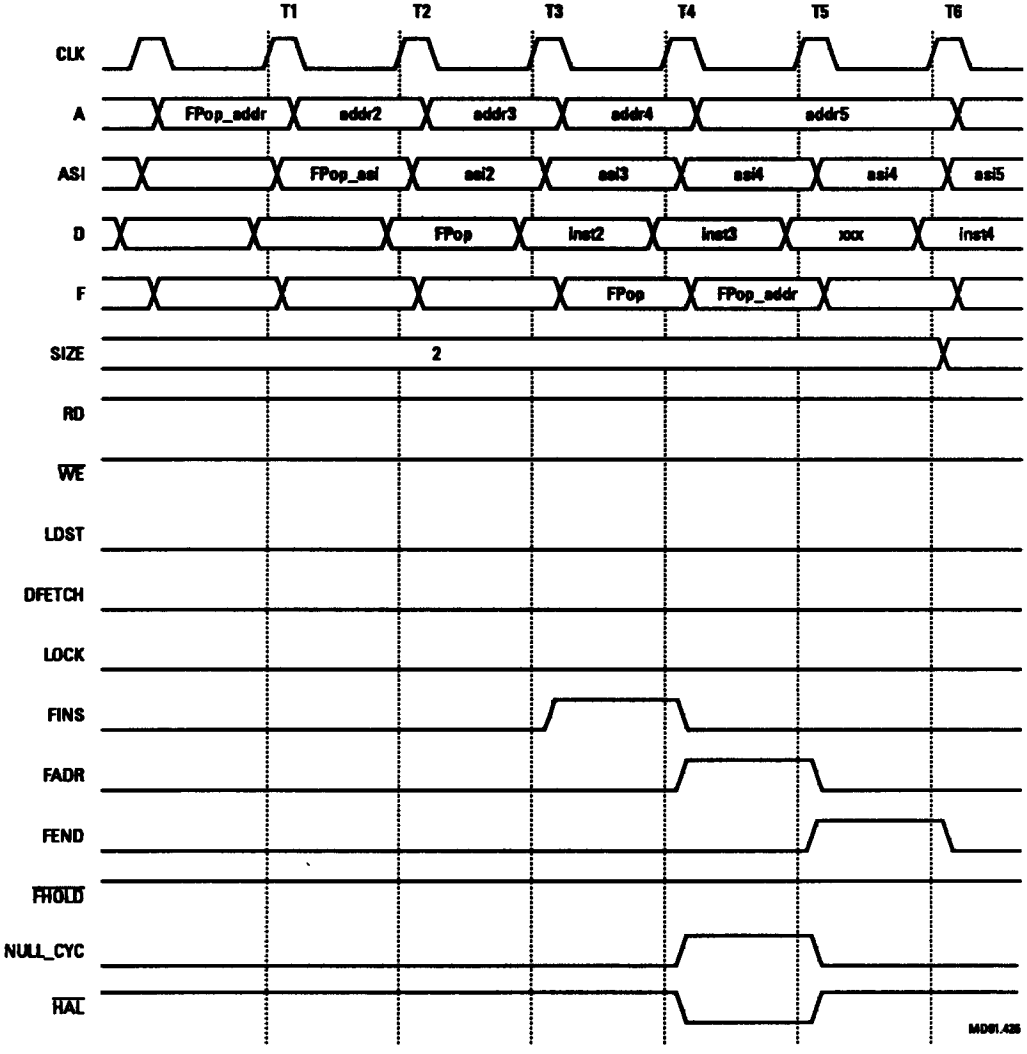
The IU fetches and decodes FPops, then sends them to the FPU over the floating-point bus (F[31:0]). The IU also provides the FPU with control signals to indicate that an FPop is decoded. During an FPop, the IU places the instruction on the floating-point bus during the execute cycle and puts the instruction address on the floating-point bus during the write cycle.

If the FPU cannot execute the current floating-point instruction, it informs the IU by asserting $\overline{\text{FHOLD}}$. This occurrence can happen under the following conditions:

- When a store FSR instruction starts execution and FPops are pending in the floating-point queue. In this case, the FPU detects the condition and asserts FHOLD. The store FSR instruction must wait until all pending FPops complete execution.
- When an FPop is issued and either a resource or an operand dependency exists between the present FPop and one or more of the previously fetched instructions.
- When a branch on floating-point condition (FBfcc) starts executing while the floating-point conditions are not ready. This condition occurs when one of the previously fetched instructions is a floating-point compare (FCMP) that the FPU has not yet completed.

Figure 2.15 shows the timing for a floating-point operation.

Figure 2.15
Floating-Point
Operation Timing



2.8 Specifications

This section presents the L64801's electrical and mechanical characteristics. This section is divided into three subsections:

- AC Timing (page 2-44)
- Electrical Requirements (page 2-47)
- Packaging (page 2-50)

AC Timing

This subsection describes the AC timing specifications of the L64801 IU. The test conditions for the AC timing values are VDD = 5 V ± 5% and TA = 0°C to 70°C. Table 2.5 lists the AC timing specifications. Figure 2.16 through Figure 2.20 illustrate the AC timing waveforms.

Table 2.5 Timing Specifications

Parameter	*Description*	*Min*	*Max*	*Units*
1	System Clock Cycle Time	50		ns
2	System Clock Rise/Fall Times		3	ns
3	System Clock High Duration	15		ns
4	System Clock Low Duration	15		ns
5	$\overline{\text{RESET}}$ Active Time	10		ns
6	Address Valid Delay from CLK Rising	5	44	ns
7	ASI Valid Delay from CLK Rising	5	32	ns
8	Read Data Setup before CLK Rising	5		ns
9	Read Data Hold after CLK Rising	5		ns
10	Write Data Valid from CLK Rising	5	32	ns
11	Write Data Turn Off from CLK	5		ns
12	SIZE Valid Delay from CLK Rising	5	20	ns
13	RD Valid Delay from CLK Rising	5	20	ns
14	$\overline{\text{WE}}$ Valid Delay from CLK Rising	5	21	ns
15	LDST Valid Delay from CLK Rising	5	20	ns
16	NULL_CYC Valid Delay from CLK Rising	5	41	ns
17	$\overline{\text{MHOLD}}$ Valid to NULL_CYC	5	22	ns
18	$\overline{\text{IH\_NULL}}$ Valid to NULL_CYC	5	14	ns
19	$\overline{\text{HAL}}$ Valid Delay from CLK Rising	5	36	ns
20	$\overline{\text{MHOLD}}$ Valid to $\overline{\text{HAL}}$	4	20	ns
21	LOCK Valid Delay from CLK Rising	5	21	ns
22	DFETCH Valid Delay from CLK Rising	5	32	ns
23	$\overline{\text{MDS}}$ Setup before CLK Falling	27		ns
24	$\overline{\text{MDS}}$ Hold after CLK Rising	0		ns
25	$\overline{\text{MHOLD}}$, $\overline{\text{SHOLD}}$, $\overline{\text{BHOLD}}$ Setup before CLK Rising	27		ns
26	$\overline{\text{MHOLD}}$, $\overline{\text{SHOLD}}$, $\overline{\text{BHOLD}}$ Hold after CLK Rising	0		ns
27	$\overline{\text{MHOLD}}$, $\overline{\text{SHOLD}}$, $\overline{\text{BHOLD}}$ Setup before CLK Falling	9		ns

Continued on next page.

Table 2.5 Timing Specifications, continued

Parameter	Description	Min	Max	Units
28	$\overline{\text{MHOLD}}$, $\overline{\text{SHOLD}}$, $\overline{\text{BHOLD}}$ Hold after CLK Falling	0		ns
29	FCC Setup before CLK Rising	5		ns
30	FCC Hold after CLK Rising	0		ns
31	FCCV Setup before CLK Rising	3		ns
32	FCCV Hold after CLK Rising	0		ns
33	$\overline{\text{FHOLD}}$ Setup before CLK Rising	2		ns
34	$\overline{\text{FHOLD}}$ Hold after CLK Rising	1		ns
35	$\overline{\text{FEXC}}$ Setup before CLK Rising	2		ns
36	$\overline{\text{FEXC}}$ Hold after CLK Rising	2		ns
37	F Valid Delay after CLK Rising	5	43	ns
38	FINS Valid Delay after CLK Rising	5	30	ns
39	FADR Valid Delay after CLK Rising	5	29	ns
40	FEND Valid Delay after CLK Rising	5	29	ns
41	FLUSH Valid Delay after CLK Rising	5	25	ns
42	FXACK Valid Delay after CLK Rising	5	29	ns
43	$\overline{\text{TC}}$ Setup before CLK Rising	12		ns
44	$\overline{\text{TC}}$ Hold after CLK Rising	0		ns
45	IRL Setup before CLK Rising	18		ns
46	IRL Hold after CLK Rising	2		ns
47	$\overline{\text{RESET}}$ Setup before CLK Rising	2		ns
48	$\overline{\text{RESET}}$ Hold after CLK Rising	2		ns
49	$\overline{\text{ERROR}}$ Valid Delay after CLK Rising	5	23	ns
50	Address Drivers Off/On after $\overline{\text{AOE}}$	4	19	ns
51	ASI Drivers Off/On after $\overline{\text{ASIOE}}$	4	16	ns
52	$\overline{\text{WE}}$ Driver Off/On after $\overline{\text{ASIOE}}$	4	16	ns
53	RD Driver Off/On after $\overline{\text{ASIOE}}$	4	16	ns
54	LDST Driver Off/On after $\overline{\text{ASIOE}}$	4	16	ns
55	Data Bus Drivers Off/On after $\overline{\text{DOE}}$	4	25	ns

Figure 2.16 Clock Timing

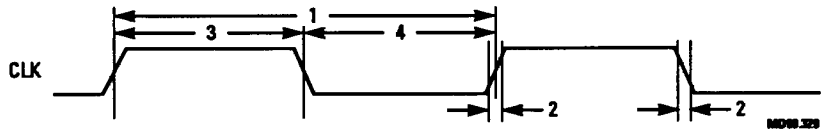


Figure 2.17
Data Bus Timing

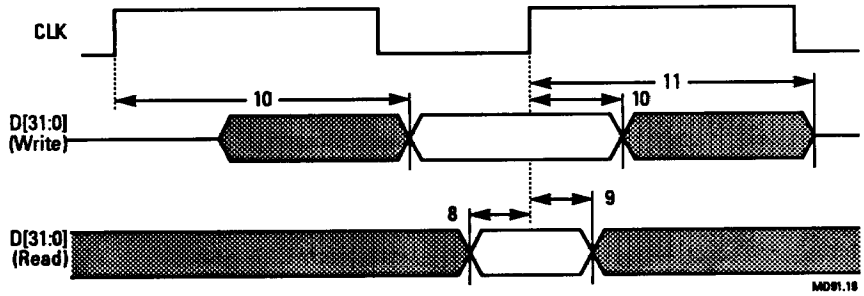
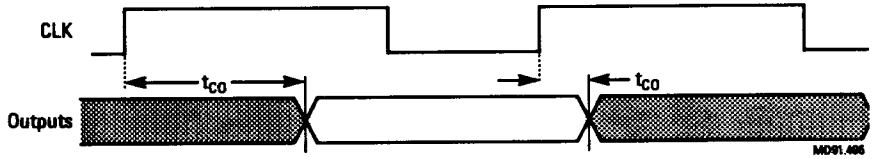
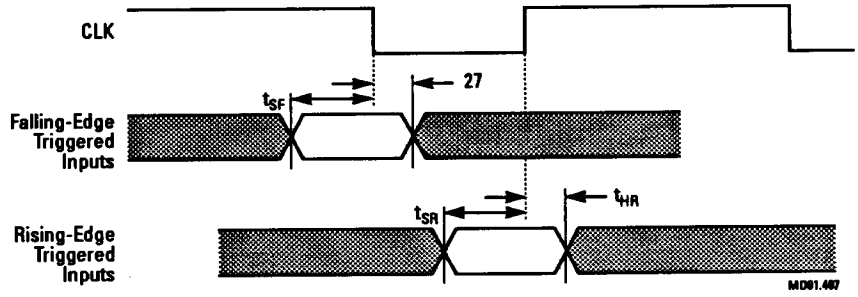


Figure 2.18
Clock-to-Output Timing



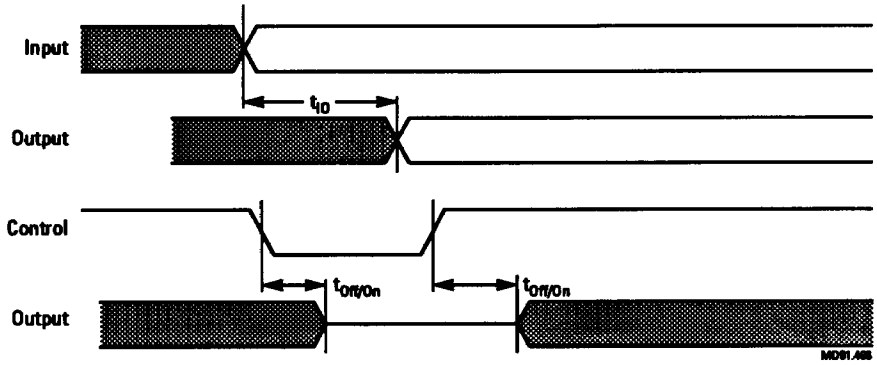
1. Clock-to-output time t_{CO} applies to parameters 6, 7, 11 through 15, 18, 20, 21, 36 through 41, and 48.

Figure 2.19
Input-to-Clock Timing



1. Setup time t_{SF} applies to parameters 22 and 26.
2. Setup time t_{SR} applies to parameters 24, 28, 30, 32, 34, 42, 44, and 46.
3. Hold time t_{HR} applies to parameters 23, 25, 29, 31, 33, 35, 43, and 45.

Figure 2.20
Input-to-Output Timing



1. I/O time t_{IO} applies to parameters 16, 17, and 19.
2. Driver off/on time $t_{Off/On}$ applies to parameters 49 through 54.

Electrical Requirements

This subsection specifies the electrical requirements for the L64801. Five tables list electrical data in the following categories:

- Absolute Maximum Ratings (Table 2.6)
- Recommended Operating Conditions (Table 2.7)
- Capacitance (Table 2.8)
- DC Characteristics (Table 2.9)
- Pin Description Summary (Table 2.10)

Table 2.6
Absolute Maximum Ratings

Symbol	*Parameter*	*Limits*[1]	*Units*
V_{DD}	DC Supply	-0.3 to +7.0	V
V_{IN}	Input Voltage	-0.3 to V_{DD} + 0.3	V
I_{IN}	DC Input Current	±10	mA
T_{STG}	Storage Temperature Range (Ceramic)	-65 to +150	°C
	Storage Temperature Range (Plastic)	-40 to +125	°C

1. Referenced to V_{SS}

Table 2.7
Recommended Operating Conditions

Symbol	*Parameter*	*Limits*	*Units*
V_{DD}	DC Supply	4.75 to 5.25	V
T_A	Ambient Temperature	0 to +70	°C

Table 2.8
Capacitance

Symbol	*Parameter*	*Condition*	*Min*	*Typ*	*Max*	*Units*
C_{IN}	Input Capacitance[1]	V_{DD} = 5 V ± 5%, f = 1 MHz, T_A = 25°C			5	pF
C_{OUT}	Output Capacitance[2]	V_{DD} = 5 V ± 5%, f = 1 MHz, T_A = 25°C			15	pF

1. Not applicable to assigned bidirectional buffer (excluding package).
2. Output using single buffer structure (excluding package).

Table 2.9
DC Characteristics

Symbol	*Parameter*	*Condition*[1]	*Min*	*Typ*	*Max*	*Units*
V_{IH}	Voltage Input High	V_{DD} = MIN	2.0			V
V_{IL}	Voltage Input Low	V_{DD} = MIN			0.8	V
V_{OH}	Voltage Output High	I_{OH} = -4.0 mA	2.4			V
V_{OL}	Voltage Output Low	I_{OL} = 4.0 mA			0.4	V
I_{IL}	Input Leakage Current	V_{DD} = MAX, V_{IN} = 0 V	-100		-2	µA
		V_{DD} = MAX, V_{IN} = 3.5 V	-100		-2	µA
I_{OZ}	Current 3-State Output Leakage	V_{DD} = MAX, V_{OUT} = 0 to V_{DD}	-10		10	µA
I_{OZU}	Current 3-State Data Bus	V_{DD} = MAX, V_{OUT} = 0 to 3.5 V	-100		-2	µA
I_{OSP}	Current P-Channel, Output Short Circuit	V_{DD} = MAX, V_{OUT} = 0	-100		-5	mA
I_{OSN}	Current N-Channel, Output Short Circuit	V_{DD} = MAX, V_{OUT} = V_{DD}	15		130	mA
I_{DD}	Quiescent Supply Current	V_{IN} = V_{DD} or V_{SS}, V_{DD} = MAX			2	mA
I_{CC}	Dynamic Supply Current	V_{DD} = MAX, f = 20 MHz			200	mA

1. Specified at V_{DD} equals 5 V ± 5% at ambient temperature over the specified range.

Table 2.10
Pin Description Summary

Mnemonic	*Description*	*Type*	*Drive (mA)*	*Active*
A[31:0]	IU Address [31:0]	Output	–	
$\overline{\text{AOE}}$	Address Output Enable	Input	–	Low
ASI[7:0]	Address Space Identifier [7:0]	3-State Output	4	
$\overline{\text{ASIOE}}$	Address Space Identifier Output Enable	Input	–	Low
$\overline{\text{BHOLD}}$	Bus Hold	Input	–	Low
CLK	Clock	Input	–	
D[31:0]	IU Data Bus [31:0]	3-State Bidirectional	4	
DFETCH	Data Fetch	Output	4	High
$\overline{\text{DOE}}$	Data Output Enable	Input	–	Low
$\overline{\text{ERROR}}$	Error	Output	4	Low
F[31:0]	Floating-Point Bus [31:0]	Output	4	
FADR	Floating-Point Address	Output	4	High
FCC[1:0]	Floating-Point Condition Codes	Input	–	
FCCV	Floating-Point Condition Code Valid	Input	–	High
FEND	End Floating-Point Instruction	Output	4	High
$\overline{\text{FEXC}}$	Floating-Point Exception	Input	–	Low
$\overline{\text{FHOLD}}$	Floating-Point Hold	Input	–	Low
FINS	Floating-Point Instruction	Output	4	High
FLUSH	Floating-Point Instruction Flush	Output	4	High
$\overline{\text{FP}}$	Floating-Point Present	Input w/Pullup	–	Low
FXACK	Floating-Point Exception Acknowledge	Output	4	High
$\overline{\text{HAL}}$	Hold Address Latch	Output	4	Low
$\overline{\text{IH\_NULL}}$	Null Cycle Reset	Input	–	Low
IRL[3:0]	Interrupt Level [3:0]	Input	–	
LDST	Load/Store Cycle	3-State Output	4	High
LOCK	Bus Lock Request	Output	4	High
$\overline{\text{MDS}}$	Memory Data Strobe	Input	–	Low
$\overline{\text{MEXC}}$	Memory Exception	Input	–	Low
$\overline{\text{MHOLDA}}$	Hold from Memory	Input	–	Low
$\overline{\text{MHOLDB}}$	Hold from Memory	Input	–	Low
$\overline{\text{MHOLDC}}$	Hold from Memory	Input	–	Low
NULL_CYC	Null Cycle	3-State Output	4	High
RD	Read Cycle	3-State Output	4	High
$\overline{\text{RESET}}$	Reset	Input	–	Low
$\overline{\text{SHOLD}}$	Hold from Memory	Input	–	Low
SIZE[1:0]	Size [1:0]	3-State Output	4	
$\overline{\text{TC}}$	Trap Condition	Input	–	Low
$\overline{\text{WE}}$	Write Enable	3-State Output	4	Low

Packaging

The L64801 IU is available in either a 160-pin PQFP, a 179-pin PPGA, or a 179-pin CPGA. Figure 2.21 and Figure 2.22 show the pinout and mechanical dimensions for the 160-pin PQFP package. Figure 2.23 shows the pinout for the 179-pin packages. Figure 2.24 shows the mechanical dimensions for the 179-pin CPGA package.

Figure 2.21
L64801 160-pin PQFP
Pinout

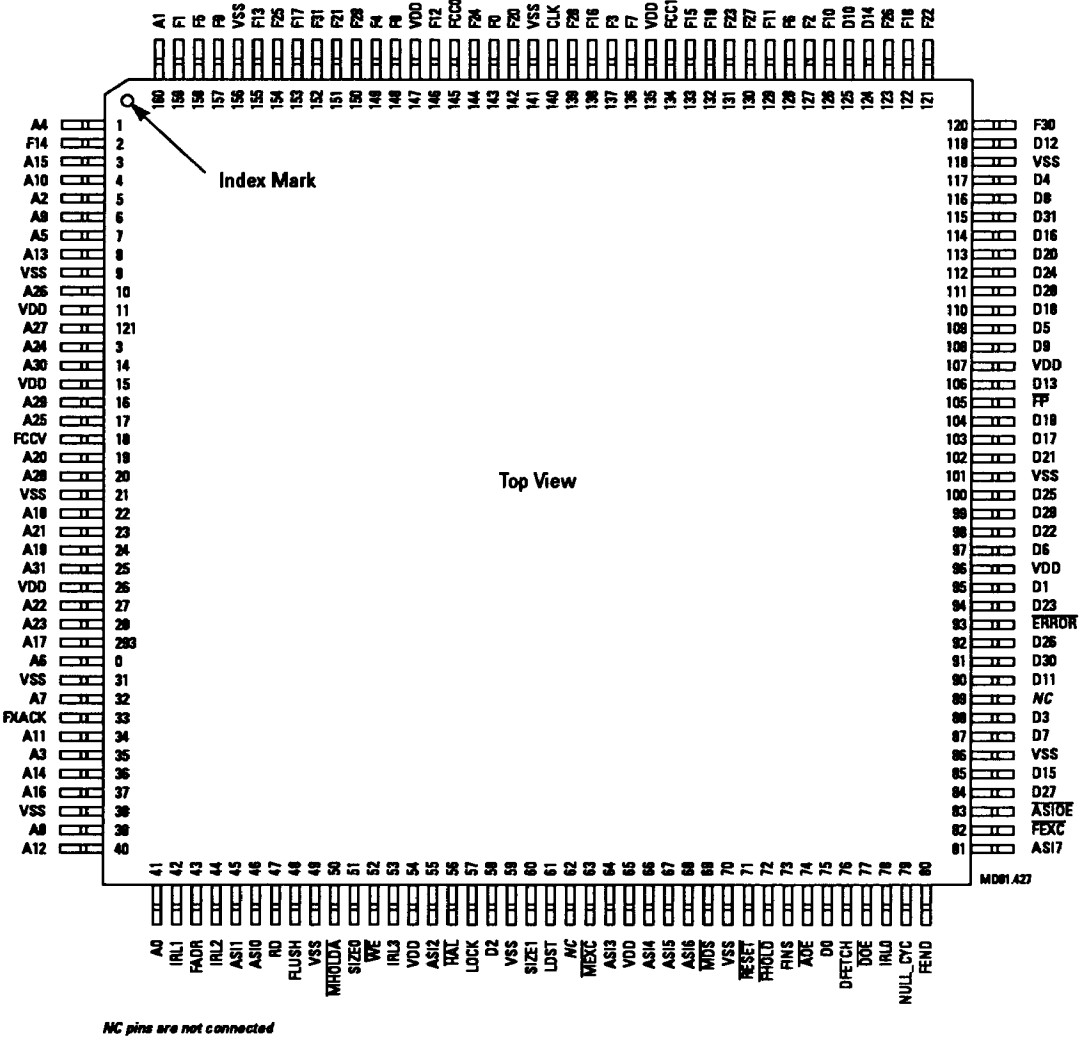
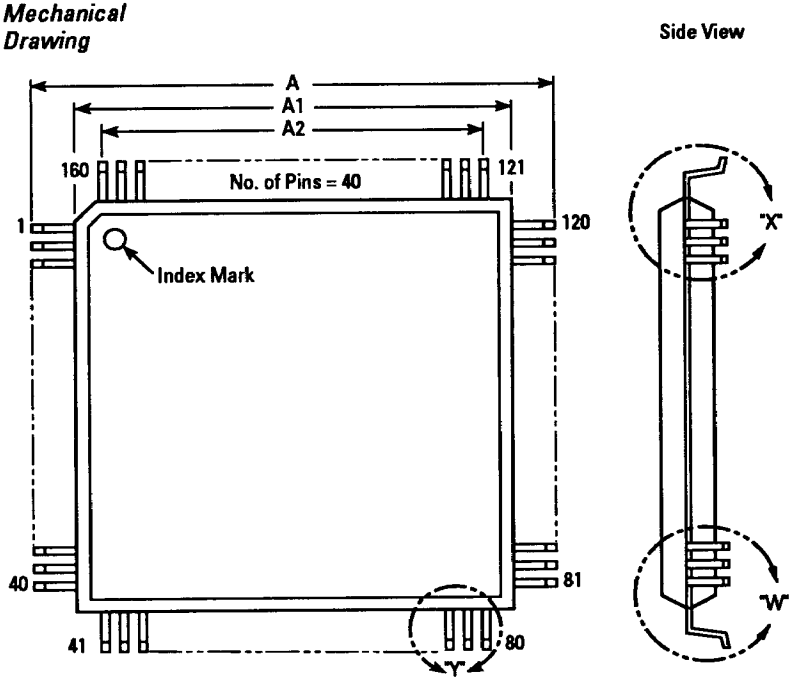
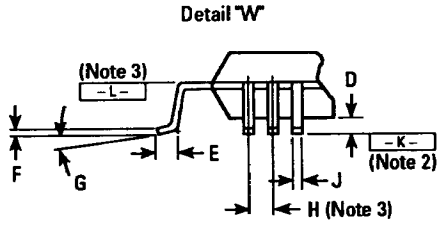


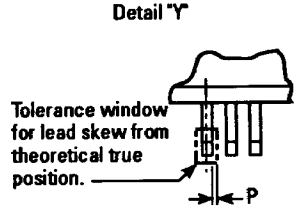
Figure 2.22
L64801 160-pin PQFP
Mechanical
Drawing

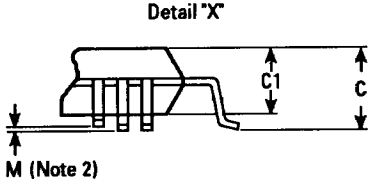


Dimension		Millimeters (Inches)
A	Min	31.60 Sq (1.244)
	Max	32.40 Sq (1.276)
A1	Min	27.90 Sq (1.098)
	Max	28.10 Sq (1.106)
A2	Ref	25.35 Sq (0.998)
C	Max	3.94 (0.155)
C1	Max	3.55 (0.140)
D	Max	0.030 (0.012)
E	Min	0.061 (0.024)
	Max	1.00 (0.039)
F	Min	0.10 (0.004)
	Max	0.25 (0.010)
G	Min	0 degrees
	Max	10 degrees
H	Nom	0.65 (0.026)
J	Min	0.25 (0.010)
	Max	0.35 (0.014)
M	Max	0.10 (0.004)
P	Max	0.05 (0.002)

Notes:
1. Controlling dimension – mm.
2. Coplanarity of all leads shall be within .010 mm (difference between the highest and lowest lead with seating plane –K– as reference).
3. Lead pitch determined at –L–.
4. Drawing is not to scale.

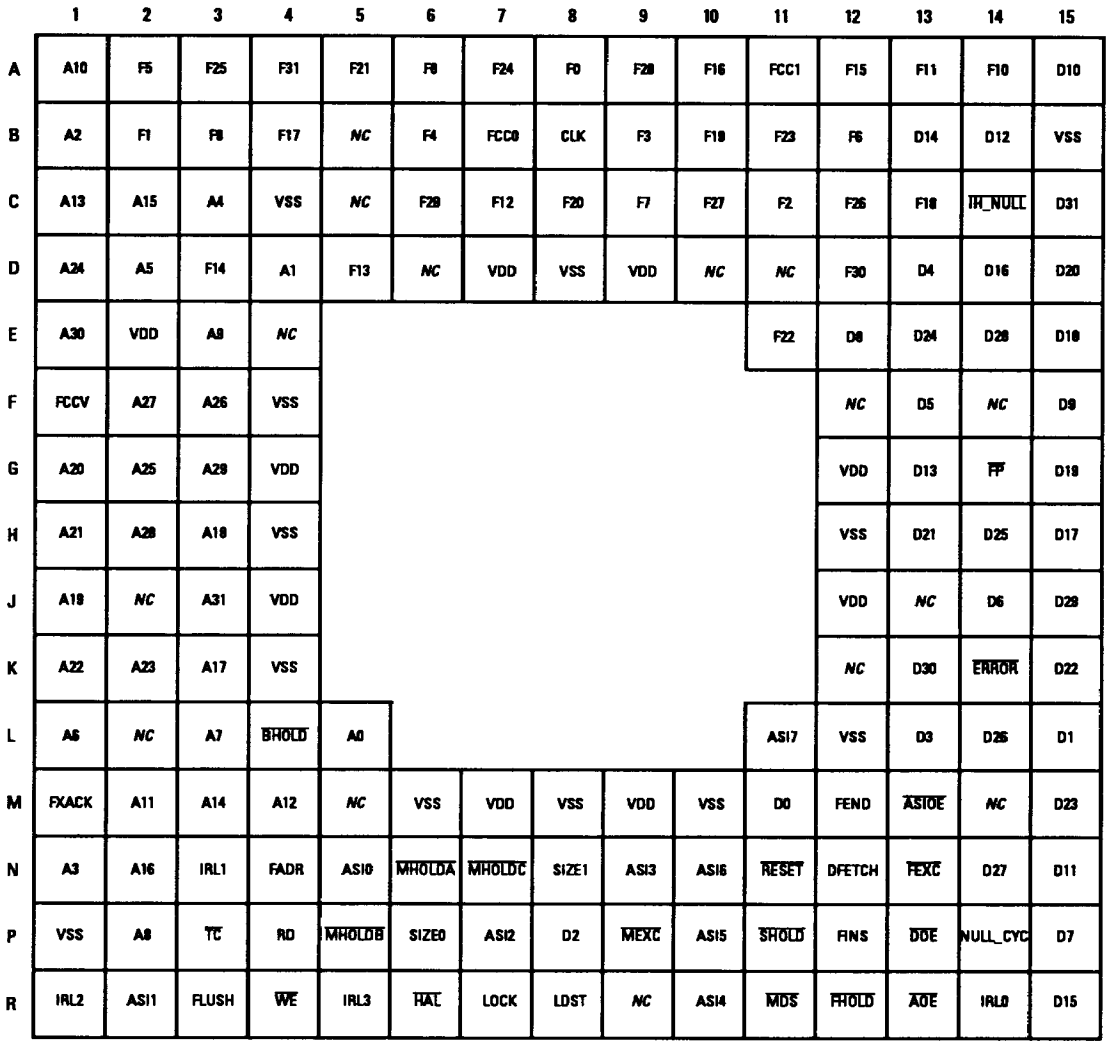






MD00.13E

Figure 2.23
L64801 179-pin
Pinout

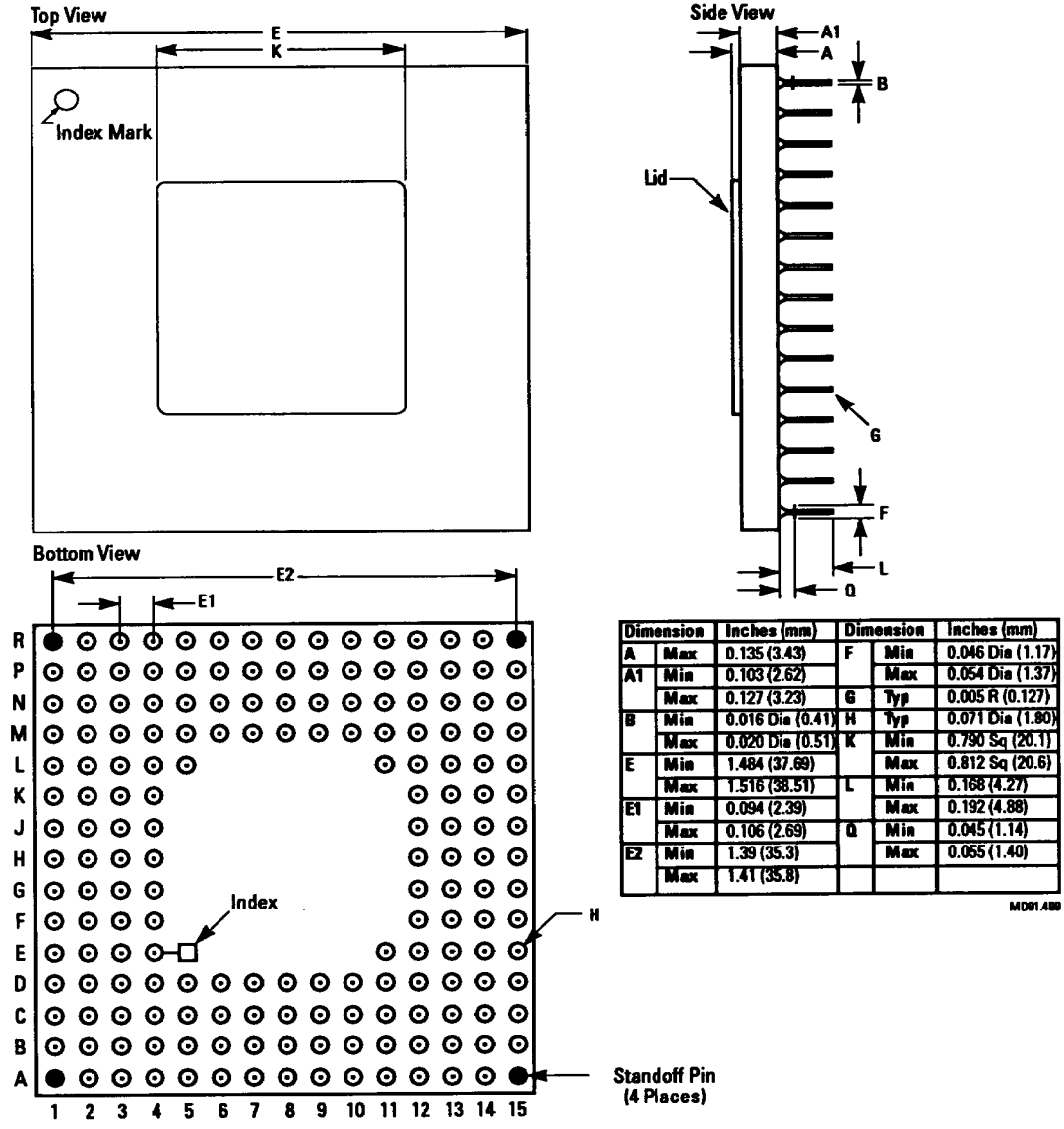


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	A10	F5	F25	F31	F21	F8	F24	F0	F28	F16	FCC1	F15	F11	F10	D10
B	A2	F1	F9	F17	*NC*	F4	FCC0	CLK	F3	F19	F23	F6	D14	D12	VSS
C	A13	A15	A4	VSS	*NC*	F29	F12	F20	F7	F27	F2	F26	F18	IR_NULL	D31
D	A24	A5	F14	A1	F13	*NC*	VDD	VSS	VDD	*NC*	*NC*	F30	D4	D16	D20
E	A30	VDD	A9	*NC*							F22	D8	D24	D28	D19
F	FCCV	A27	A26	VSS								*NC*	D5	*NC*	D9
G	A20	A25	A29	VDD								VDD	D13	FP	D18
H	A21	A28	A18	VSS								VSS	D21	D25	D17
J	A19	*NC*	A31	VDD								VDD	*NC*	D6	D29
K	A22	A23	A17	VSS								*NC*	D30	ERROR	D22
L	A6	*NC*	A7	BHOLD	A0						ASI7	VSS	D3	D26	D1
M	FXACK	A11	A14	A12	*NC*	VSS	VDD	VSS	VDD	VSS	D0	FEND	ASIOE	*NC*	D23
N	A3	A16	IRL1	FADR	ASI0	MHOLDA	MHOLDC	SIZE1	ASI3	ASI6	RESET	DFETCH	FEXC	D27	D11
P	VSS	A8	TC	RD	MHOLDB	SIZE0	ASI2	D2	MEXC	ASI5	SHOLD	FINS	DOE	NULL_CYC	D7
R	IRL2	ASI1	FLUSH	WE	IRL3	HAL	LOCK	LDST	*NC*	ASI4	MDS	FHOLD	AOE	IRL0	D15

NC pins are not connected

MD81.428

Figure 2.24
L64801 179-pin CPGA
Mechanical
Drawing



Dimension		Inches (mm)	Dimension		Inches (mm)
A	Max	0.135 (3.43)	F	Min	0.046 Dia (1.17)
A1	Min	0.103 (2.62)		Max	0.054 Dia (1.37)
	Max	0.127 (3.23)	G	Typ	0.005 R (0.127)
B	Min	0.016 Dia (0.41)	H	Typ	0.071 Dia (1.80)
	Max	0.020 Dia (0.51)	K	Min	0.790 Sq (20.1)
E	Min	1.484 (37.69)		Max	0.812 Sq (20.6)
	Max	1.516 (38.51)	L	Min	0.168 (4.27)
E1	Min	0.094 (2.39)		Max	0.192 (4.88)
	Max	0.106 (2.69)	Q	Min	0.045 (1.14)
E2	Min	1.39 (35.3)		Max	0.055 (1.40)
	Max	1.41 (35.8)			

MD01.4B0