Chapter 3

Instruction Formats (I)

3.1 Alpha Registers

Each Alpha processor has a set of registers that hold the current processor state. If an Alpha system contains multiple Alpha processors, there are multiple per-processor sets of these registers.

3.1.1 Program Counter

The Program Counter (PC) is a special register that addresses the instruction stream. As each instruction is decoded, the PC is advanced to the next sequential instruction. This is referred to as the *updated PC*. Any instruction that uses the value of the PC will use the updated PC. The PC includes only bits <63:2> with bits <1:0> treated as RAZ/IGN. This quantity is a long-word-aligned byte address. The PC is an implied operand on conditional branch and subroutine jump instructions. The PC is not accessible as an integer register.

3.1.2 Integer Registers

There are 32 integer registers (R0 through R31), each 64 bits wide.

Register R31 is assigned special meaning by the Alpha architecture. When R31 is specified as a register source operand, a zero-valued operand is supplied.

For all cases except the Unconditional Branch and Jump instructions, results of an instruction that specifies R31 as a destination operand are discarded. Also, it is UNPREDICTABLE whether the other destination operands (implicit and explicit) are changed by the instruction. It is implementation dependent to what extent the instruction is actually executed once it has been fetched. An exception is never signaled for a load that specifies R31 as a destination operation. For all other operations, it is UNPREDICTABLE whether exceptions are signaled during the execution of such an instruction. Note, however, that exceptions associated with the instruction fetch of such an instruction are always signaled.

Implementation note:

As described in Appendix A, certain load instructions to an R31 destination are the preferred method for performing a cache block prefetch.

There are some interesting cases involving R31 as a destination:

- STx_C R31,disp(Rb)

  Although this might seem like a good way to zero out a shared location and reset the lock_flag, this instruction causes the lock_flag and virtual location {Rbv + SEXT(disp)} to become UNPREDICTABLE.

- LDx_L R31,disp(Rb)

  This instruction produces no useful result since it causes both lock_flag and locked_physical_address to become UNPREDICTABLE.

Unconditional Branch (BR and BSR) and Jump (JMP, JSR, RET, and JSR_COROUTINE) instructions, when R31 is specified as the Ra operand, execute normally and update the PC with the target virtual address. Of course, no PC value can be saved in R31.

3.1.3 Floating-Point Registers

There are 32 floating-point registers (F0 through F31), each 64 bits wide.

When F31 is specified as a register source operand, a true zero-valued operand is supplied. See Section 4.7.3 for a definition of true zero.

Results of an instruction that specifies F31 as a destination operand are discarded and it is UNPREDICTABLE whether the other destination operands (implicit and explicit) are changed by the instruction. In this case, it is implementation-dependent to what extent the instruction is actually executed once it has been fetched.

A memory management exception or alignment exception is never signaled for a load that specifies F31 as a destination register. It is UNPREDICTABLE whether a floating-point disabled exception can be signaled by a load that specifies F31 as a destination register. For all other instructions that specify F31 as an output operand, it is UNPREDICTABLE whether exceptions are signaled during the execution of such an instruction. Note, however, that exceptions associated with the instruction fetch of such an instruction are always signaled.

Implementation note:

As described in Appendix A, certain load instructions to an F31 destination are the preferred method for signalling a cache block prefetch.

A floating-point instruction that operates on single-precision data reads all bits <63:0> of the source floating-point register. A floating-point instruction that produces a single-precision result writes all bits <63:0> of the destination floating-point register.

3.1.4 Lock Registers

There are two per-processor registers associated with the LDx_L and STx_C instructions, the lock_flag and the locked_physical_address register. The use of these registers is described in Section 4.2.

3.1.5 Processor Cycle Counter (PCC) Register

The PCC register consists of two 32-bit fields. The low-order 32 bits (PCC<31:0>) are an unsigned wrapping counter, PCC_CNT. The high-order 32 bits (PCC<63:32>), PCC_OFF, are operating system dependent in their implementation.

PCC_CNT is the base clock register for measuring time intervals and is suitable for timing intervals on the order of nanoseconds.

PCC_CNT increments once per N CPU cycles, where N is an implementation-specific integer in the range 1..16. The cycle counter frequency is the number of times the processor cycle counter gets incremented per second. The integer count wraps to 0 from a count of FFFF $FFFF_{16}$. The counter wraps no more frequently than 1.5 times the implementation's interval clock interrupt period (which is two thirds of the interval clock interrupt frequency), which guarantees that an interrupt occurs before PCC _CNT overflows twice.

PCC_OFF need not contain a value related to time and could contain all zeros in a simple implementation. However, if PCC_OFF is used to calculate a per-process or per-thread cycle count, it must contain a value that, when added to PCC_CNT, returns the total PCC register count for that process or thread, modulo 2**32.

Implementation Note:

OpenVMS, Tru64 UNIX, and Alpha Linux supply a per-thread value in PCC_OFF.

PCC is required on all implementations. It is required for every processor, and each processor on a multiprocessor system has its own private, independent PCC.

The PCC is read by the RPCC instruction. See Section 4.11.9.

3.1.6 Optional Registers

Some Alpha implementations may include optional memory prefetch or VAX compatibility processor registers.

3.1.6.1 Memory Prefetch Registers

If the prefetch instructions FETCH and FETCH_M are implemented, an implementation will include two sets of state prefetch registers used by those instructions. The use of these registers is described in Section 4.11. These registers are not directly accessible by software and are listed for completeness.

3.1.6.2 VAX Compatibility Register

The VAX compatibility instructions RC and RS include the intr_flag register, as described in Section 4.12.

3.2 Notation

The notation used to describe the operation of each instruction is given as a sequence of control and assignment statements in an ALGOL-like syntax.

3.2.1 Operand Notation

Tables 3–1, 3–2, and 3–3 list the notation for the operands, the operand values, and the other expression operands.

Table 3–1 Operand Notation

Notation	Meaning
Ra	An integer register operand in the Ra field of the instruction
Rb	An integer register operand in the Rb field of the instruction
#b	An integer literal operand in the Rb field of the instruction
Rc	An integer register operand in the Rc field of the instruction
Fa	A floating-point register operand in the Ra field of the instruction
Fb	A floating-point register operand in the Rb field of the instruction
Fc	A floating-point register operand in the Rc field of the instruction

Table 3–2 Operand Value Notation

Notation	Meaning
Rav	The value of the Ra operand. This is the contents of register Ra.
Rbv	The value of the Rb operand. This could be the contents of register Rb, or a zero-extended 8-bit literal in the case of an Operate format instruction.
Fav	The value of the floating-point Fa operand. This is the contents of register Fa.
Fbv	The value of the floating-point Fb operand. This is the contents of register Fb.

Table 3–3 Expression Operand Notation

Notation	Meaning
IPR_x	Contents of Internal Processor Register x
IPR_SP[mode]	Contents of the per-mode stack pointer selected by mode
PC	Updated PC value
Rn	Contents of integer register n
Fn	Contents of floating-point register n
X[m]	Element m of array X

3.2.2 Instruction Operand Notation

The notation used to describe instruction operands follows from the operand specifier notation used in the *VAX Architecture Standard*. Instruction operands are described as follows:

```
<name>.<access type><data type>
```

3.2.2.1 Operand Name Notation

Specifies the instruction field (Ra, Rb, Rc, or disp) and register type of the operand (integer or floating). It can be one of the following:

Table 3–4 Operand Name Notation

Name	Meaning
disp	The displacement field of the instruction
fnc	The PALcode function field of the instruction
Ra	An integer register operand in the Ra field of the instruction
Rb	An integer register operand in the Rb field of the instruction
#b	An integer literal operand in the Rb field of the instruction
Rc	An integer register operand in the Rc field of the instruction
Fa	A floating-point register operand in the Ra field of the instruction
Fb	A floating-point register operand in the Rb field of the instruction
Fc	A floating-point register operand in the Rc field of the instruction

3.2.2.2 Operand Access Type Notation

A letter that denotes the operand access type:

Table 3–5 Operand Access Type Notation

Access Type	Meaning
a	The operand is used in an address calculation to form an effective address. The data type code that follows indicates the units of addressability (or scale factor) applied to this operand when the instruction is decoded. For example: ".al" means scale by 4 (longwords) to get byte units (used in branch displacements); ".ab" means the operand is already in byte units (used in load/store instructions).
i	The operand is an immediate literal in the instruction.
r	The operand is read only.
m	The operand is both read and written.
w	The operand is write only.

3.2.2.3 Operand Data Type Notation

A letter that denotes the data type of the operand:

Table 3–6 Operand Data Type Notation

Data Type	Meaning
b	Byte
f	F_floating
g	G_floating
l	Longword
q	Quadword
s	IEEE single floating (S_floating)
t	IEEE double floating (T_floating)
w	Word
x	The data type is specified by the instruction

3.2.3 Operators

Table 3–7 describes the operators:

Table 3–7 Operators

Operator	Meaning
!	Comment delimiter.
+	Addition.
-	Subtraction.
*	Signed multiplication.
*U	Unsigned multiplication.
**	Exponentiation (left argument raised to right argument).
/	Division.
←	Replacement.
||	Bit concatenation.
{}	Indicates explicit operator precedence.
(x)	Contents of memory location whose address is x.
x <m:n>	Contents of bit field of x defined by bits n through m.
x <m>	M'th bit of x.
ACCESS(x,y)	Accessibility of the location whose address is x using the access mode y. Returns a Boolean value TRUE if the address is accessible, else FALSE.

Table 3–7 Operators (Continued)

Operator	Meaning
AND	Logical product.
ARITH_RIGHT_SHIFT(x,y)	Arithmetic right shift of first operand by the second operand. Y is an unsigned shift value. Bit 63, the sign bit, is copied into vacated bit positions and shifted out bits are discarded.
BYTE_ZAP(x,y)	X is a quadword, y is an 8-bit vector in which each bit corresponds to a byte of the result. The y bit to x byte correspondence is y <n> ↔ x <8n+7:8n>. This correspondence also exists between y and the result. For each bit of y from n = 0 to 7, if y <n> is 0 then byte <n> of x is copied to byte <n> of result, and if y <n> is 1 then byte <n> of result is forced to all zeros.

CASE	The CASE construct selects one of several actions based on the value of its argument. The form of a case is:
	`CASE argument OF`
	`  argvalue1: action_1`
	`  argvalue2: action_2`
	`  ...`
	`  argvaluen:action_n`
	`  [otherwise: default_action]`
	`ENDCASE`
	If the value of argument is argvalue1 then action_1 is executed; if argument = argvalue2, then action_2 is executed, and so forth.
	Once a single action is executed, the code stream breaks to the END-CASE (there is an implicit break as in Pascal). Each action may nonetheless be a sequence of pseudocode operations, one operation per line.
	Optionally, the last argvalue may be the atom ‘otherwise’. The associated default action will be taken if none of the other argvalues match the argument.

DIV	Integer division (truncates).
LEFT_SHIFT(x,y)	Logical left shift of first operand by the second operand.Y is an unsigned shift value. Zeros are moved into the vacated bit positions, and shifted out bits are discarded.
LOAD_LOCKED	The processor records the target physical address in a per-processor locked_physical_address register and sets the per-processor lock_flag.
lg	Log to the base 2.
MAP_x	F_float or S_float memory-to-register exponent mapping function.
MAXS(x,y)	Returns the larger of x and y, with x and y interpreted as signed integers.

Table 3–7 Operators (Continued)

Operator	Meaning
MAXU(x,y)	Returns the larger of x and y, with x and y interpreted as unsigned integers.
MINS(x,y)	Returns the smaller of x and y, with x and y interpreted as signed integers.
MINU(x,y)	Returns the smaller of x and y, with x and y interpreted as unsigned integers.
x MOD y	x modulo y.
NOT	Logical (ones) complement.
OR	Logical sum.
PHYSICAL_ADDRESS	Translation of a virtual address.
PRIORITY_ENCODE	Returns the bit position of most significant set bit, interpreting its argument as a positive integer (=int(lg(x))). For example: `priority_encode( 255 ) = 7`
Relational Operators:	

Operator	Meaning
LT	Less than signed
LTU	Less than unsigned
LE	Less or equal signed
LEU	Less or equal unsigned
EQ	Equal signed and unsigned
NE	Not equal signed and unsigned
GE	Greater or equal signed
GEU	Greater or equal unsigned
GT	Greater signed
GTU	Greater unsigned
LBC	Low bit clear
LBS	Low bit signed

RIGHT_SHIFT(x,y)	Logical right shift of first operand by the second operand. Y is an unsigned shift value. Zeros are moved into vacated bit positions, and shifted out bits are discarded.
SEXT(x)	X is sign-extended to the required size.
STORE_CONDITIONAL	If the lock_flag is set, then do the indicated store and clear the lock_flag.

Table 3–7 Operators (Continued)

Operator	Meaning
TEST(x,cond)	The contents of register x are tested for branch condition (cond) true. TEST returns a Boolean value TRUE if x bears the specified relation to 0, else FALSE is returned. Integer and floating test conditions are drawn from the preceding list of relational operators.
XOR	Logical difference.
ZEXT(x)	X is zero-extended to the required size.

3.2.4 Notation Conventions

The following conventions are used:

- Only operands that appear on the left side of a replacement operator are modified.
- No operator precedence is assumed other than that replacement (←) has the lowest precedence. Explicit precedence is indicated by the use of "{}".
- All arithmetic, logical, and relational operators are defined in the context of their operands. For example, "+" applied to G_floating operands means a G_floating add, whereas "+" applied to quadword operands is an integer add. Similarly, "LT" is a G_floating comparison when applied to G_floating operands and an integer comparison when applied to quadword operands.

3.3 Instruction Formats

There are five basic Alpha instruction formats:

- Memory
- Branch
- Operate
- Floating-point Operate
- PALcode

All instruction formats are 32 bits long with a 6-bit major opcode field in bits <31:26> of the instruction.

Any unused register field (Ra, Rb, Fa, Fb) of an instruction must be set to a value of 31.

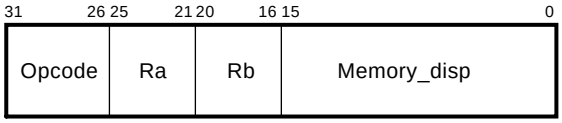
Software Note:

There are several instructions, each formatted as a memory instruction, that do not use the Ra and/or Rb fields. These instructions are: Memory Barrier, Fetch, Fetch_M, Read Process Cycle Counter, Read and Clear, Read and Set, and Trap Barrier.

3.3.1 Memory Instruction Format

The Memory format is used to transfer data between registers and memory, to load an effective address, and for subroutine jumps. It has the format shown in Figure 3–1.

Figure 3–1: Memory Instruction Format



A Memory format instruction contains a 6-bit opcode field, two 5-bit register address fields, Ra and Rb, and a 16-bit signed displacement field.

The displacement field is a byte offset. It is sign-extended and added to the contents of register Rb to form a virtual address. Overflow is ignored in this calculation.

The virtual address is used as a memory load/store address or a result value, depending on the specific instruction. The virtual address (va) is computed as follows for all memory format instructions except the load address high (LDAH):

va ← {Rbv + SEXT(Memory_disp)}

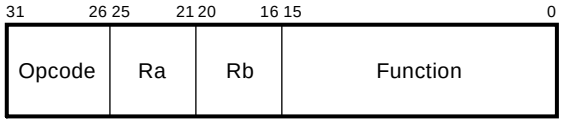
For LDAH the virtual address (va) is computed as follows:

va ← {Rbv + SEXT(Memory_disp*65536)}

3.3.1.1 Memory Format Instructions with a Function Code

Memory format instructions with a function code replace the memory displacement field in the memory instruction format with a function code that designates a set of miscellaneous instructions. The format is shown in Figure 3–2.

Figure 3–2: Memory Instruction with Function Code Format



The memory instruction with function code format contains a 6-bit opcode field and a 16-bit function field. Unused function codes produce UNPREDICTABLE but not UNDEFINED results; they are not security holes.

There are two fields, Ra and Rb. The usage of those fields depends on the instruction. See Section 4.11.

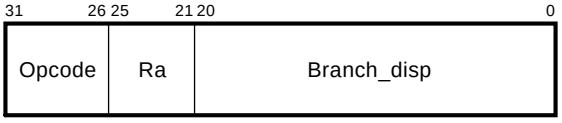
3.3.1.2 Memory Format Jump Instructions

For computed branch instructions (CALL, RET, JMP, JSR_COROUTINE) the displacement field is used to provide branch-prediction hints as described in Section 4.3.

3.3.2 Branch Instruction Format

The Branch format is used for conditional branch instructions and for PC-relative subroutine jumps. It has the format shown in Figure 3–3.

Figure 3–3: Branch Instruction Format



A Branch format instruction contains a 6-bit opcode field, one 5-bit register address field (Ra), and a 21-bit signed displacement field.

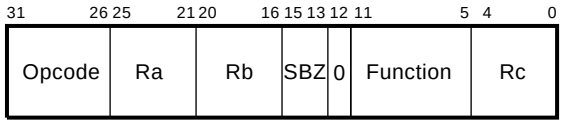
The displacement is treated as a longword offset. This means it is shifted left two bits (to address a longword boundary), sign-extended to 64 bits, and added to the updated PC to form the target virtual address. Overflow is ignored in this calculation. The target virtual address (va) is computed as follows:

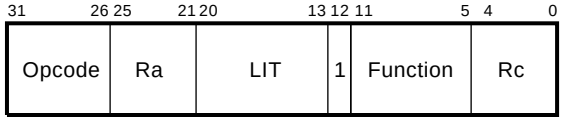
```
va ← PC + {4*SEXT(Branch_disp)}
```

3.3.3 Operate Instruction Format

The Operate format is used for instructions that perform integer register to integer register operations. The Operate format allows the specification of one destination operand and two source operands. One of the source operands can be a literal constant. The Operate format in Figure 3–4 shows the two cases when bit <12> of the instruction is 0 and 1.

Figure 3–4: Operate Instruction Format





An Operate format instruction contains a 6-bit opcode field and a 7-bit function code field. Unused function codes for opcodes defined as reserved in the Version 5 Alpha architecture specification (May 1992) produce an illegal instruction trap. Those opcodes are 01, 02, 03, 04, 05, 06, 07, 0A, 0C, 0D, 0E, 14, 19, 1B, 1C, 1D, 1E, and 1F. For other opcodes, unused function codes produce UNPREDICTABLE but not UNDEFINED results; they are not security holes.

There are three operand fields, Ra, Rb, and Rc.

The Ra field specifies a source operand. Symbolically, the integer Rav operand is formed as follows:

```
IF inst<25:21> EQ 31 THEN
    Rav ← 0
ELSE
    Rav ← Ra
END
```

The Rb field specifies a source operand. Integer operands can specify a literal or an integer register using bit <12> of the instruction.

If bit <12> of the instruction is 0, the Rb field specifies a source register operand.

If bit <12> of the instruction is 1, an 8-bit zero-extended literal constant is formed by bits <20:13> of the instruction. The literal is interpreted as a positive integer between 0 and 255 and is zero-extended to 64 bits. Symbolically, the integer Rbv operand is formed as follows:

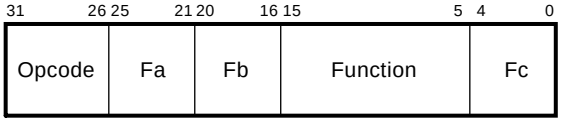
```
IF inst <12> EQ 1 THEN
    Rbv ← ZEXT(inst<20:13>)
ELSE
    IF inst <20:16> EQ 31 THEN
         Rbv ← 0
    ELSE
         Rbv ← Rb
    END
END
```

The Rc field specifies a destination operand.

3.3.4 Floating-Point Operate Instruction Format

The Floating-point Operate format is used for instructions that perform floating-point register to floating-point register operations. The Floating-point Operate format allows the specification of one destination operand and two source operands. The Floating-point Operate format is shown in Figure 3–5.

Figure 3–5: Floating-Point Operate Instruction Format



A Floating-point Operate format instruction contains a 6-bit opcode field and an 11-bit function field. Unused function codes for those opcodes defined as reserved in the Version 5 Alpha architecture specification (May 1992) produce an illegal instruction trap. Those opcodes are 01, 02, 03, 04, 05, 06, 07, 14, 19, 1C, 1B, 1D, 1E, and 1F. For other opcodes, unused function codes produce UNPREDICTABLE but not UNDEFINED results; they are not security holes.

There are three operand fields, Fa, Fb, and Fc. Each operand field specifies either an integer or floating-point operand as defined by the instruction.

The Fa field specifies a source operand. Symbolically, the Fav operand is formed as follows:

```
IF inst<25:21> EQ 31 THEN
    Fav ← 0
ELSE
    Fav ← Fa
END
```

The Fb field specifies a source operand. Symbolically, the Fbv operand is formed as follows:

```
IF inst<20:16> EQ 31 THEN
    Fbv ← 0
ELSE
    Fbv ← Fb
END
```

Note:

Neither Fa nor Fb can be a literal in Floating-point Operate instructions.

The Fc field specifies a destination operand.

3.3.4.1 Floating-Point Convert Instructions

Floating-point Convert instructions use a subset of the Floating-point Operate format and perform register-to-register conversion operations. The Fb operand specifies the source; the Fa field must be F31.

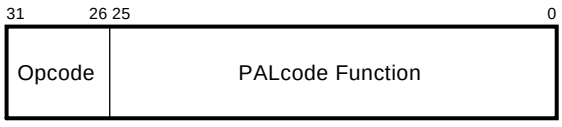
3.3.4.2 Floating-Point/Integer Register Moves

Instructions that move data between a floating-point register file and an integer register file are a subset of the Floating-point Operate format. The unused source field must be 31.

3.3.5 PALcode Instruction Format

The Privileged Architecture Library (PALcode) format is used to specify extended processor functions. It has the format shown in Figure 3–6.

Figure 3–6: PALcode Instruction Format



The 26-bit PALcode function field specifies the operation. The source and destination operands for PALcode instructions are supplied in fixed registers that are specified in the individual instruction descriptions.

An opcode of zero and a PALcode function of zero specify the HALT instruction.