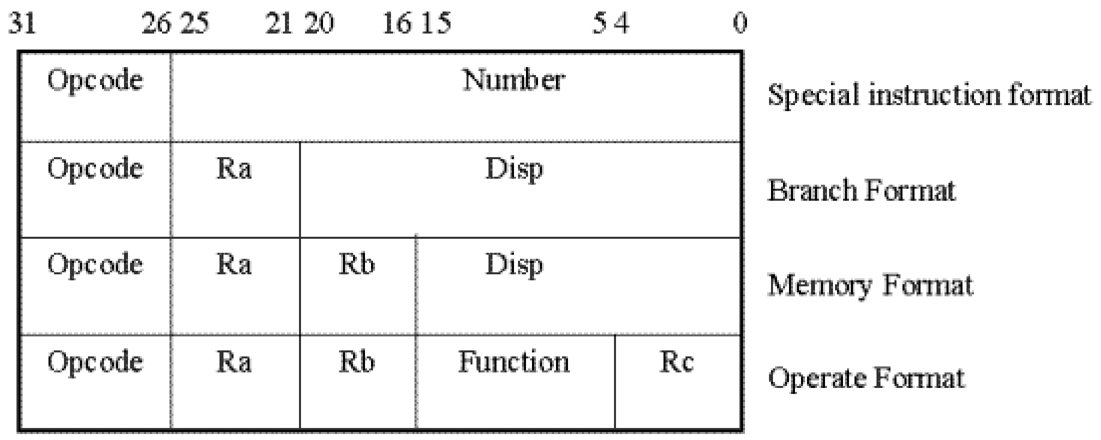
Integer operate instructions

Integer operate instructions are used to perform operations on values in integer registers.

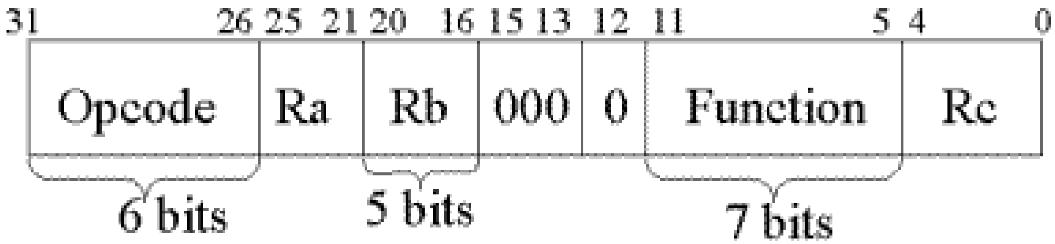
Instructions Format

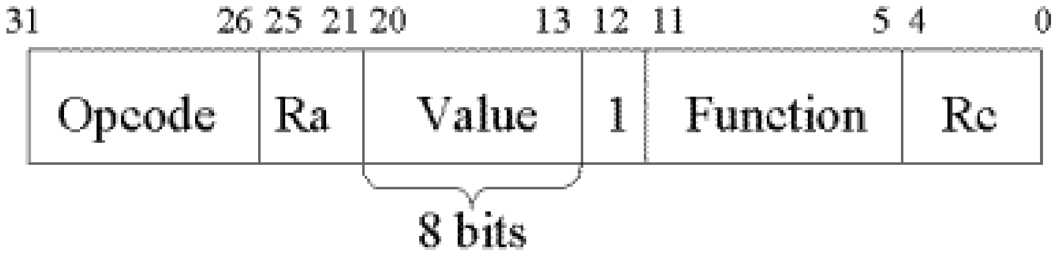
Alpha instructions are all 32 bits long. Each instruction has a 6 bits opcode field. There are four major instruction formats. We are going to study three of them, i.e. operate format, memory format and branch format.



The operate format is for instructions performing integer register to integer register operations such as addq, subq. The operate format allows the specification of one destination operand and two source operands. One of the source operands can be a literal value. The two formats are distinguished by bit 12. If one of the source operand is a literal value, bit 12 is set to 1; otherwise, bit 12 is 0. It can be seen that each register field consists of 5 bits. This is because there are 32 integer registers. Thus, 5 bits is sufficient to hold the number denoting a register. In the diagrams below, Ra and Rb are the source registers' fields and Rc is the destination register's field. The literal is interpreted as a positive integer between 0 and 255 and is zero-extended to 64 bits.

Operate instruction format





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.

Example:
If the 8-bit literal value is 0x40 the zero-extended 64-bit value is used for the operation:

- 0x40 –> 0x0000000000000040

If the 8-bit literal value is 0x90 it is view as a positive value and is zero-extended to a 64-bit value for the required operation

Rule:

Contents of registers must be written as 64 bits integer
Literal value involved in integer operates instructions must be written as 8 bits value and zero-extended to 64 bits value when performing an operation.

<u>Addition, multiplication, division</u>

Addition

addq, *subq* and *mulq* instructions can have two or three operands. The first operand must be a register. The other operands can be either a litteral value or a register. For example:

```
addq $T0, $T1, $T2    $T2 = $T0+$T1
(addq s_reg1,s_reg2,  d_reg)

addq $T0, $T1           $T0 = $T0+$T1
(addq d_reg/s_reg1, s_reg2)

addq $T0, 0x1, $T1            $T1 = $T0+0x1
(addq s_reg, val, d_reg)

addq $T0, 0x1           $T0 = $T0+0x1
(addq d_reg/s_reg, val)
```

Substraction

```
subq  $T0, $T1, $T2            $T2 = $T0-$T1
(subqs_reg1, s_reg2, d_reg)

subq  $T0, $T1           $T0 = $T0-$T1
(subqd_reg/s_reg1, s_reg2)

subq  $T0, 0x1, $T1            $T1 = $T0-0x1
(subqs_reg, val, d_reg)

subq  $T0, 0x1    $T0 = $T0-0x1
(subqd_reg/s_reg, val)
```

Multiplication

```
mulq $T0, $T1, $T2          $T2 = $T0*$T1
(mulq      s_reg1, s_reg2, d_reg)

mulq $T0, $T1          $T0 = $T0*$T1
(mulq      d_reg/s_reg1, s_reg2)

mulq $T0, 0x1, $T1          $T1 = $T0*0x1
(mulq      s_reg, val, d_reg)

mulq $T0, 0x1          $T0 = $T0*0x1
(mulq      d_reg/s_reg, val)
```

Alpha does not have integer division instructions but the simulator does.
The macro-instruction set includes integer division instructions.

- Assembler replaces the division instructions with a procedure to carry out the division instructions.
- Integer division can be programmed quite easily:
    - § 7 divided by 2: 7=3*2+1

Example

Suppose we have the following values in registers.

```
$t0 0x0000000000000000
$t1 0x0000000000000037
$t9 0x0000000000000093
$10 0xffffffffffffff93
```

How will the registers change after executing the instructions?

```
subq $t0, 1;
addq $t1, 0x45;
addq $t9, 0x94;
addq $t10, 0x94;
mulq $t1, 0x4;
```

Boolean computations

Some “integer operate” instructions for performing Boolean computations are:
“and” (&), “bic” (bit clear & ~), “bis” (bit set) or “or” (|), “eqv” (equivalent) or “xornot” (exclusive or not) ^~, “ornot” | ~, “xor” (exclusive or ^).
Example, “bic $1, $2, $3;” means
intReg[3] = intReg[1] & ~ intReg[2];

Boolean instructions tables:

T0	T1	AND	AND NOT	OR	ORNOT	XOR	XORNOT
0	0	0	0	0	1	0	1
0	1	0	0	1	0	1	0
1	0	0	1	1	1	1	0
1	1	1	0	1	1	0	1

not instruction calculates the complement of each bit of a value. The instruction can have one or two operands. In the two-operand format, the complement of each of the first operand is stored in the corresponding of the second operand. In the one-operand format, each bit of the operand is complemented.

```
not   s_reg, d_reg                 not   $T0, $T1
T0 = 0x0000000000000876
 T1 = 0xfffffffffffff789                 T0 unchanged

not   value, d_reg                 not   0xf1, $T0
T0 = 0x

not   d_reg/s_reg                  not   $T0
```

T0 = 0x0..0123 **T0 =**

and instruction is normally used to clear selected bits in a register. It can also be used to check if a specified bit is set. The instruction can have two or three operands. The first operand must be a register. The instruction carries out a bit-wise AND to the first two operands.

and s_reg1, s_reg2, d_reg and $T0, $T1, $T2
T0 = 0x000000000000008f T1 = 0x00000000000000e9
T2 = 0x0000000000000089

and s_reg, value, d_reg and $T0, 0xe1, $T1
T0 = 0x0000000000000078 **T1 = 0x0000000000000060**

and d_reg/s_reg1, s_reg2
and $T0, $T1

and d_reg/s_reg, value and $T0, 0xa2
T0 = 0x0000000000000083
T0 = 0x0000000000000082

or instruction is used to set certain bits in a register to one while leaving others unchanged. The instruction can have two or three operands. The first operand must be a register. The instruction carries out a bit-wise OR to the first two operands.

or s_reg1, s_reg2, d_reg or $T0, $T1, $T2
T0 = 0x000000000000008f T1 = 0x00000000000000e9
T2 = 0x00000000000000ef

or s_reg, value, d_reg or $T0, 0xe1, $T1
T0 = 0x0000000000000078 **T1 = 0x00000000000000f9**

or d_reg/s_reg1, s_reg2 or $T0, $T1
T0 = 0 T1 = 0x0000000000000012 **T1 = 0x0000000000000012**

or d_reg/s_reg, value or $T0, 0xa2
`T0 = 0x0000000000000083` **`T0 = 0x0000… 000a3`**

Shift Instructions

Three shift instructions: “ sll” (shift left logical), “ sra” (shift right arithmetic), and “srl” (shift right logical), corresponding to <<, >> and >>>.

- Used to shift the bit patterns left and right.
- The shift logical instructions fill the vacated bits with 0
- The shift right arithmetic instruction fills the vacated bits with the sign bit.

These instructions can be used to extract fields out of a bit pattern, and interpret them as either unsigned or signed numbers.

- A cheap way to multiply or divide by a power of 2.

sll shifts the contents of a register to the left. The instruction can have two or three operands. The first operand must be a register. The first operand holds the value to be shifted. The second operand indicates the number of bits to be shifted. In the two operands format, the first register holds the result of the operation. In the three operands format, the third operand stores the result. The bits vacated by the shift are filled with 0s.

sll s_reg1, s_reg2, d_reg sll $T0, $T1, $T2

- T0 = 0x0…08f T1 = x0…02 **T2 = 0x0…023c**
- If s_reg2 > 63 or < 0, shifted by s_reg2 AND 63

sll s_reg, value, d_reg sll $T0, 0x1, $T1

- T0 = 0x8000000000000078 **T1 = 0x0…0f0**

sll d_reg/s_reg1, s_reg2 sll $T0, $T1

- T0 = 0x0…012 T1 = 0x0…02 **T0 = 0x0…048**

sll d_reg/s_reg, value sll $T0, 0x1

- T0 = 0xc000000000000003 **T0 = 0x8000000000000006** *srl* shifts the contents of a register to the right. The instruction can have two or three operands. The first operand must be a register. The first operand holds the value to be shifted. The second operand indicates the number of bits to be shifted. In the two operands format, the first register holds the result of the operation. In the three operands format, the third operand stores the result. The bits vacated by the shift are filled with 0s.

srl s_reg1, s_reg2, d_reg srl $T0, $T1, $T2

- T0 = 0x0…08f T1 = 0x0…02 **T2 = 0x0…023**
- If s_reg2 > 63 or < 0, shifted by s_reg2 AND 63

srl s_reg, value, d_reg srl $T0, 0x1, $T1

- T0 = 0x8000000000000078 **T1 = 0x400000000000003c**

srl d_reg/s_reg1, s_reg2 srl $T0, $T1

- T0 = 0x0…012 T1 = 0x0…02 **T0 = 0x0…04**

srl d_reg/s_reg, value srl $T0, 0x2
T0 = 0xc000000000000003 **T0 = 0x3000000000000000**

sra is similar to srl. The only difference between the two instructions is the sign bit is used to set the vacated bits in sra.

sra s_reg1, s_reg2, d_reg sra $T0, $T1, $T2
T0 = 0x8000..00f T1 = 0x2 **T2 = 0xe00…03**
If s_reg2 > 63 or < 0, shifted by s_reg2 AND 63

sra s_reg, value, d_reg sra $T0, 0x1, $T1
T0 = 0x8000000000000078 **T1 = 0xc00000000000003d**

sra d_reg/s_reg1, s_reg2 sra $T0, $T1
T0 = 0x0…012 T2 = 0x2 **T1 = 0x0…04**

sra d_reg/s_reg, value sra $T0, 0x2
T0=0xc000000000000003 **T0=0xf000000000000000**