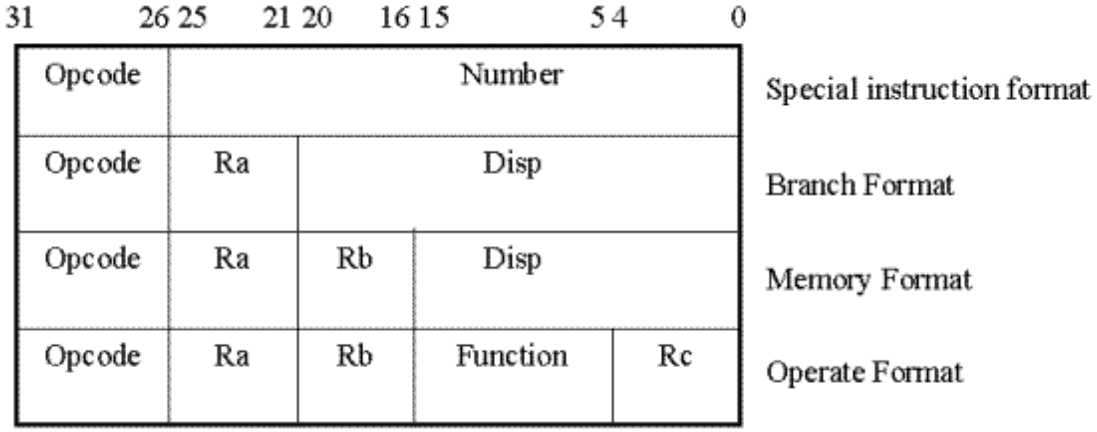
Memory Instructions

Memory instructions are used to transfer data between registers and memory, to load an effective address, and for subroutine jumps.
Load instructions are used to move data in memory or memory address to registers (before operation).
Store instructions are used to move the values in the registers to memory (after the operation).
They have the following 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 an effective address.

- Overflow is ignored in this calculation.
- 16-bit constant displacement between –32,768 and 32,767

To operate on memory values, we must first load the source data from memory, perform the computation, and then store the result back in memory.
Load and **store** instructions have the form:

opcode $regA, displacement($regB);

Effective address

The effective address is the 64-bit twos-complement sum of the contents of the index register and the sign-extended offset (otherwise called displacement). It provides the address where the data has to be stored to or to be loaded from.

Integer load instructions load an appropriate number of bytes starting at the specified (effective) address, and store them in intReg[regA].
For example:
ldq (load quadword) loads the 8 bytes corresponding to a quadword, starting at the memory address, into register intReg[regA].
ldbu (load byte unsigned) loads a single byte from the memory address, into the low byte of register intReg[regA], making the high 7 bytes zero.

Integer store instructions store an appropriate number of bytes from register intReg[regA] to the memory starting at the specified address.
For example:
stq (store quadword) stores all 8 bytes from register intReg[regA] corresponding to a quadword, into memory at a starting address given by the effective address.
stb (store byte) stores the low byte of register intReg[regA], into memory at a starting address given by the effective address.

The default on the Alpha is to store data in memory in little endian format.

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

$$EA \leftarrow \{Rbv + SEXT(Memory\_disp)\}$$

Value hold by the register Rb

Sign-extended value

For LDAH, the effective address is computed as:

$$EA \leftarrow \{Rbv + SEXT(Memory\_disp * 65536)\}$$

Sll 16 bits

Load signed values from memory to registers

ldX instructions are used to move signed values to registers.
X indicates the type of data to be moved. X is either b (byte) or w (word) or l (longword) or q (quadword).
ldX has two operands.
The first operand must be a register, holds the value, which is moved from the memory.
The second operand must be either a memory label or an expression holding the address of the memory. For example,

```
ldb     $T0, ($T0)
ldb     $T1, 2($T1)
```

The sign bit of the data item is replicated to fill the significant bits of the register.

Ldx Ra, Disp(Rb)

EA = Rbv + Disp (except ldah)

`ldbu` and `ldwu` instructions are used to load a byte or a word as an unsigned value to a register.

- Only for byte and word

These instructions are similar to `ldb` and `ldw` instructions except that the sign bit of the value is not replicated to the significant bits of the registers.

Lda: Loads the destination register with the effective address of the specified data item.
Ldah: Loads the destination register with the effective address of the specified data item. In computing the effective address, the signed constant offset is multiplied by 65536 before adding to the base register. The signed constant must be in the range -32768 to 32767.

Both instructions can be seen as a move function (moving the effective address to the Register Ra). Actually MOV is a pseudo-instructions coded using lda instruction.

Assume the following directives have been used to reserve locations in the memory:

```
data{
a:   byte 0x12
b:   word 0x9876
c:   long 0x89012345
d:   quad 0x1234567890123456
}
```

Fill the memory assuming that label **a** is stored at starting address 0x100..00

label	address	content	label	address	content
a	0x10…00			0x10…08	
	0x10…01			0x10…09	
	0x10…02			0x10…0a	
	0x10…03			0x10…0b	
	0x10…04			0x10…0c	
	0x10…05			0x10…0d	
	0x10…06			0x10…0e	
	0x10…07			0x10…0f	

Assume the following directives have been used to reserve locations in the memory:

```
data{
a:   byte 0x12
b:   word 0x9876
c:   long 0x89012345
d:   quad 0x1234567890123456
}
```

Fill the memory assuming that label **a** is stored at starting address 0x100..00

label	address	contents	label	address	contents
a	0x10…00	12	d	0x10…08	56
	0x10…01	00		0x10…09	34
b	0x10…02	76		0x10…0a	12
	0x10…03	98		0x10…0b	90
c	0x10…04	45		0x10…0c	78
	0x10…05	23		0x10…0d	56
	0x10…06	01		0x10…0e	34
	0x10…07	89		0x10…0f	12

Let's suppose that:
$T4 = 0x10…00
$T5 = 0x10…02
$T6 = 0x10…04
$T7 = 0x10…08

After the execution of the following instructions, the values in the registers are as below:

```
     ldb $T0, 0($T4);
     ldw $T1, 0($T5);
     ldl $T2, 0($T6);
     ldq $T3, 0($T7);

T0   0x0000000000000012
T1   0xffffffffffff9876
T2   0xffffffff89012345
T3   0x1234567890123456
```

The sign bit is replicated

0x9876: 1001 1000 0111 0110: -2^15 + 2^12+……
0xff9876: 1111 1111 1001 1000 0111 0110:

-2^23+2^22+2^21+2^20+2^19+…+2^15+2^12+…. =
-2^22 +2^21+…… =
-2^21 +2^20+…….. =

-2^16+2^15+2^12+……. =
-2^15+2^12+……. =
0x9876

By the same way you can demonstrate that the quadword 0xff…ff9876 is equal to the word 0x9876. Using 2's representation convention you can also see that the quadword 0xff…ff9876 (which is a negative number) is equal to the word –0x678a

Assume memory label `a` is at address 0x10…00. The value of register `S0` is 0x10…00. After the execution of the following instructions, the values of the `T` registers are as below:

```
    ldb $T4, 0($S0)
    ldw $T5, 2($S0)
    ldl $T6, 4($S0)
    ldq $T7, 8($S0)

T4  0x0000000000000012
T5  0xfffffffffffff9876
T6  0xffffffff89012345
T7  0x1234567890123456
```

Assume label `a` holds the value 0x10…0. The value holds by register `S0, T2 and T3` are 0x10…00, 0x10…00, 0x10…02. After the execution of the following instructions, the values of the `T` registers are as below:

```
    ldbu $T0, 0($T2);
    ldwu $T1, 0($T3);
    ldbu $T4, 0($S0);
    ldwu $T5, 2($S0);

t0  0x0000000000000012
t1  0x0000000000009876
t4  0x0000000000000012
t5  0x0000000000009876
```

- Sign bit is not replicated

The load instructions discussed so far require the data being loaded conform to the alignment requirement specified by the instructions. In the above examples, "*ldl $T0, 0($T2);*" and "*ldq $T1, 0($T2);*" cannot be executed, since the label b does not point to an address which is longword or quadword aligned.

- Address accessed for load/store instructions must be aligned (multiple of 2, 4, 8 for, respectively ldw, ldl, ldq) with respect to the instruction involved.

- Only address need aligned, ldl $T0, 0($T2); is correct. The data item can be anything.

Assume the following directives have been used to reserve locations in the memory:

```
data{
align quad;
a:  byte 0x12
align quad;
b:  word 0x9876
align quad;
c:  long 0x89012345
align quad;
d:  quad 0x1234567890123456
}
```

Fill the memory assuming that label **a** is stored at starting address 0x1000000

	address	content		address	content		address	content		address	content
a	00			08							
	01			09							
	02			0a							
	03			0b							
	04			0c							
	05			0d							
	06			0e							
	07			0f							

Assuming that $T4 holds the value 0x1000000
Execute the following instructions:

```
    ldb $T0, 0($T4);
    ldw $T1, 8($T4);
    ldl $T2, 0x10($T4);
    ldq $T3, 24($T4);
```

```
$T0=
$T1=
$T2=
$T3=
```