Move literal values to registers

You can assign a constant value to a register using `ldiq` instruction. For example,
`"ldiq $T0, 0x123"` sets the value of `$T0` to `0x0000000000000123`.

Ldiq is also used to pass the value of a label into a register.
ldiq $T1, a;

Move address to registers

`lda` is used to pass memory address in a register rather than the contents of the memory at the address. It is doing something like:

```
(Quadword) address = displacement + intReg[ regB ];
intReg[ regA ] = address;
```

For example, assume the value stored in T0 is 0x10…00. Then, the execution of the following instruction will set the values in the T registers as below:
`lda $T2, 3($T0)`

`T2  0x1000000000000003`

The load address instruction can be seen as an *add* instruction with a constant, except that the constant is a 16 bit signed value, rather than an 8 bit unsigned value. It is often used when passing reference parameters to functions.

Store data to memory

`stX` instructions are used to move data from registers to the memory.
X indicates the type of data to be moved. X is either b (byte) or w (word) or l (longword) or q (quadword).

stX has two operands.

The first operand must be a register, which holds the value to store in memory. The second operand must be either a memory label or an expression holding the virtual address where data should be stored in memory.

stX Ra, Disp(Rb)

$Va = (Rb)_{value} + (Disp)_{sign\ extended}$

For example:

```
stb $T1, 2($T0);
```

Stores the least significant byte of $T1 at the effective address computed as the sum of the contents of $T0 and the 64-bit sign extended offset (here 0x2).

```
stw $T1, 0x32($T0);
```

Stores the 2 least significant bytes of $T1 at the effective address computed as the sum of the contents of $T0 and the 64-bit sign extended offset (here 0x32).

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

```
data {
a:   byte 0;
b:   word 0;
c:   long 0;
d:   quad 0;
     }
```

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

Assume the values in the T registers are as below:

```
T0   0x1000000000000000
T1   0x1234567890123456
T2   0x9876543210987654
T3   0xabcdef0123456789
T4   0x123456789abcdef0
```

After the execution of the following instructions, the contents of the memory becomes:

```
     stb $T1, 0($T0);
     stw $T2, 2($T0);
     stl $T3, 4($T0);
     stq $T4, 8($T0);
```

label	address	contents	label	address	contents
a	0x10…00	56	d	0x10…08	f0
	0x10…01	00		0x10…09	de
b	0x10…02	54		0x10…0a	bc
	0x10…03	76		0x10…0b	9a
c	0x10…04	89		0x10…0c	78
	0x10…05	67		0x10…0d	56
	0x10…06	45		0x10…0e	34
	0x10…07	23		0x10…0f	12

- `stX` instructions also require the starting address where data should be stored to respect the alignment requirement specified by the instruction.

"`stl $T3, 2($T0)`" and "`stq $T4, 2($T0)`" cannot be executed, since the effective address is neither longword nor quadword aligned.