Table 2.4 Example SRC Load and Store Instructions

Instruction	op	ra	rb	c1	Meaning	Addressing Mode
`ld r1, 32`	1	1	0	32	R[1] ← M[32]	Direct
`ld r22, 24(r4)`	1	22	4	24	R[22] ← M[24 + R[4]]	Displacement
`st r4, 0(r9)`	3	4	9	0	M[R[9]] ← R[4]	Register indirect
`la r7, 32`	5	7	0	32	R[7] ← 32	Immediate
`ldr r12, -48`	2	12	–	–48	R[12] ← M[PC –48]	Relative
`lar r3, 0`	6	3	–	0	R[3] ← PC	Register (!)

Example 2.2 Binary Encoding of an SRC Instruction As an example of SRC instruction encoding, let us encode the second instruction in Table 2.4, which is `ld r22, 24(r4)`. Working from the msb, the encoding will be

```
 op=1   ra=22   rb=4        c1=24
00001  10110  00100  0000000000011000 = 0x0D880018
```

■

You should verify this encoding and try several examples for yourself.

2.3.4 Arithmetic and Logic Instructions

This class of instructions uses the ALU of the SRC machine to do arithmetic, logical, and shift operations. We first cover the "1-operand" instructions `not` and `neg`.

1-Operand ALU Instructions

```
neg ra, rc ;Negate: R[ra] = –R[rc]
not ra, rc ;Not: R[ra] = R[rc]
```

These format 3 instructions take one register operand and provide one register result. The instruction `neg` (op = 15) takes the 2's complement of the contents of register R[rc] and stores it in register R[ra]. The `not` (op = 24) instruction takes the logical (1's) complement of the contents of register R[rc] and stores it in register R[ra]. All other fields in the instruction are unused.

The instructions `add` (op = 12), `sub` (op = 14), `and` (op = 20), and `or` (op = 22) are 2-operand, 1-result instructions. All must be in the general purpose registers. They are specified using format 6. Notice that the least significant 12 bits are unused, because the first 4 fields are sufficient to describe the entire operation.

2-Operand ALU Instructions

```
add ra, rb, rc ;2's complement addition: R[ra] = R[rb] + R[rc]
sub ra, rb, rc ;2's complement subtraction: R[ra] = R[rb] – R[rc]
and ra, rb, rc ;Logical AND: R[ra] = R[rb]∧R[rc]
or  ra, rb, rc ;Logical OR: R[ra] = R[rb]∨R[rc]
```