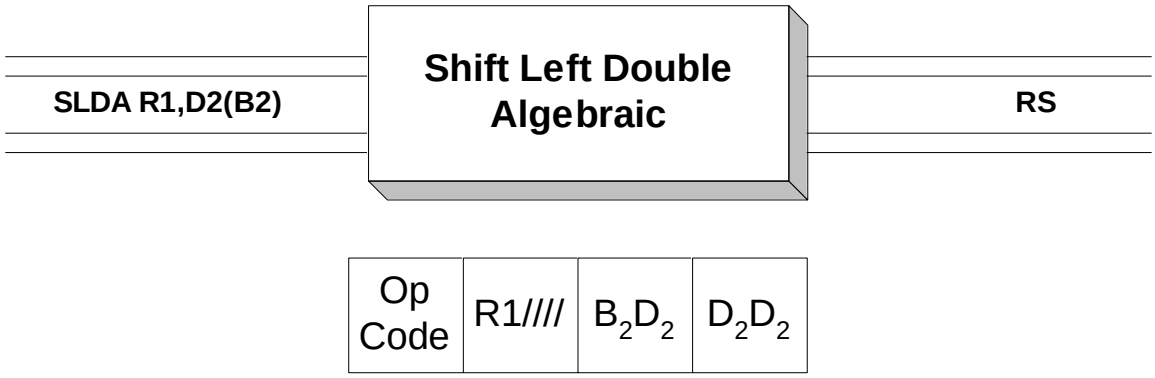



Operand 1 specifies the even register of an even-odd consecutive pair of general purpose registers. For instance R4 would represent registers 4 and 5, while R8 would represent registers 8 and 9. **SLDA** is used to shift the 64 bits in the even-odd pair as if they comprised a single register. The shift is to the left. The number of bits that are shifted is indicated by Operand 2. The second operand address is **not** used to address data; instead, the base/displacement address is computed and the rightmost 6 bits are treated as a binary integer which represents the number of bits to be shifted. We will call this value the "shift factor". This leads to two distinct ways of coding the shift factor:

1) **Directly** - The shift factor is coded as a displacement. Consider the example below.

```
SLDA  R8,5
```

In the above shift, the second operand, 5, is treated as a base/displacement address where 5 is the displacement and the base register is omitted. The effective address is 5. (See **Explicit Addressing**.) When represented as an address, the rightmost 6 bits still represent the number 5, and so the bits in registers 8 and 9 are shifted to the left by 5 bits.

2) **Indirectly** - The shift factor is placed in a register and the register is mentioned as the base register in the base/displacement address.

```
         L    R5,FACTOR    PUT SHIFT FACTOR IN REG
         SLDA  R8,0(R5)     NOW SHIFT INDIRECTLY
         ...
FACTOR   DC   F'8'          SHIFT FACTOR IS 8 BITS
```

In this case, the effective address is computed by adding the contents of base register 5 (which is 8), with the displacement of 0. The effective address is again 8, and the rightmost 6 bits of this address indicate that the shift factor is 8.

Each method has its uses. The direct method is useful in situations where the number of bits you want to shift is fixed. Coding directly allows you to look at the instruction to determine the shift factor. On the other hand, the indirect method allows the shift factor to be determined while the program is executing. If the shift factor cannot be determined until the program is running, the indirect method must be used.

When shifting algebraically, bits shifted out on the left are lost, while 0's replace bits on the right. The sign bit in Operand 1 remains fixed, preserving the sign of the integer. If one or more bits unlike the sign bit are shifted out on the left, an overflow occurs and the condition code is set to 3. This will cause a program interruption if the fixed-point overflow mask is set to 1. (See **SPM**.) Otherwise the condition code can be tested using **BO**.

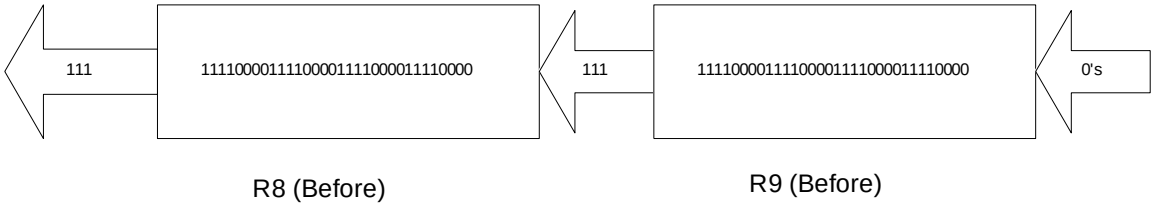
The condition code is set by this instruction in all cases:

| Condition Code | Meaning | Test With |
|---|---|---|
| 0 | Result = 0 (No overflow) | BZ, BE |
| 1 | Result < 0 (No overflow) | BL, BM |
| 2 | Result > 0 (No overflow) | BH, BP |
| 3 | Overflow | BO |

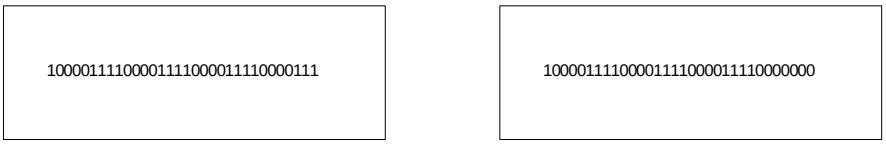
Consider the following instruction.

```
SLDA  R8,3
```

This instruction represents a left algebraic shift of registers 8 and 9 using a shift factor of 3. The shift factor has been coded directly. As a result, 3 bits, 111, are shifted out of the register on the left leaving the sign bit fixed. Vacated bit positions on the right are replace by 0's. This is illustrated in the diagram below. The condition code is set to 1, indicating that the resulting 64-bit binary integer is negative.




R8 (Before) R9 (Before)




R8 (After) R9 (After)

This instruction has an **RS** format but the 4 low-order bits of the second byte are unused.

**Examples**

**Some Unrelated SLDA's**

```
                R6 = B'11111111111111111111111111111111'
                R7 = B'00001111000011110000111100001111'

  SLDA  R6,1      R6 = B'11111111111111111111111111111110'  Cond.Code =
1
                  R7 = B'00011110000111100001111000011110'
  SLDA  R6,2      R6 = B'11111111111111111111111111111100'  Cond.Code =
1
                  R7 = B'00111100001111000011110000111100'
```

```
  SLDA  R6,3     R6 = B'11111111111111111111111111111000'  Cond.Code =
1
                 R7 = B'01111000011110000111100001111000'

  SLDA  R6,31    R6 = B'10000111100001111000011110000111'  Cond.Code =
1
                 R7 = B'10000000000000000000000000000000'

  SLDA  R6,32    R6 = B'10001111000011110000111100001111'  Cond.Code =
3
                 R7 = B'00000000000000000000000000000000'  (Overflow)


  L    R9,=F'3'
  SLDA R6,0(R9)  R6 = B'11111111111111111111111111111000'
                 R7 = B'01111000011110000111100001111000'
```

# Tips

1) Shifting a binary number to the left one digit is equivalent to multiplying it by 2. Using **SLDA**, it is a simple matter to multiply by powers of 2. For instance, to multiply by $2^5$, shift left 5 digits.