# Addressing Modes

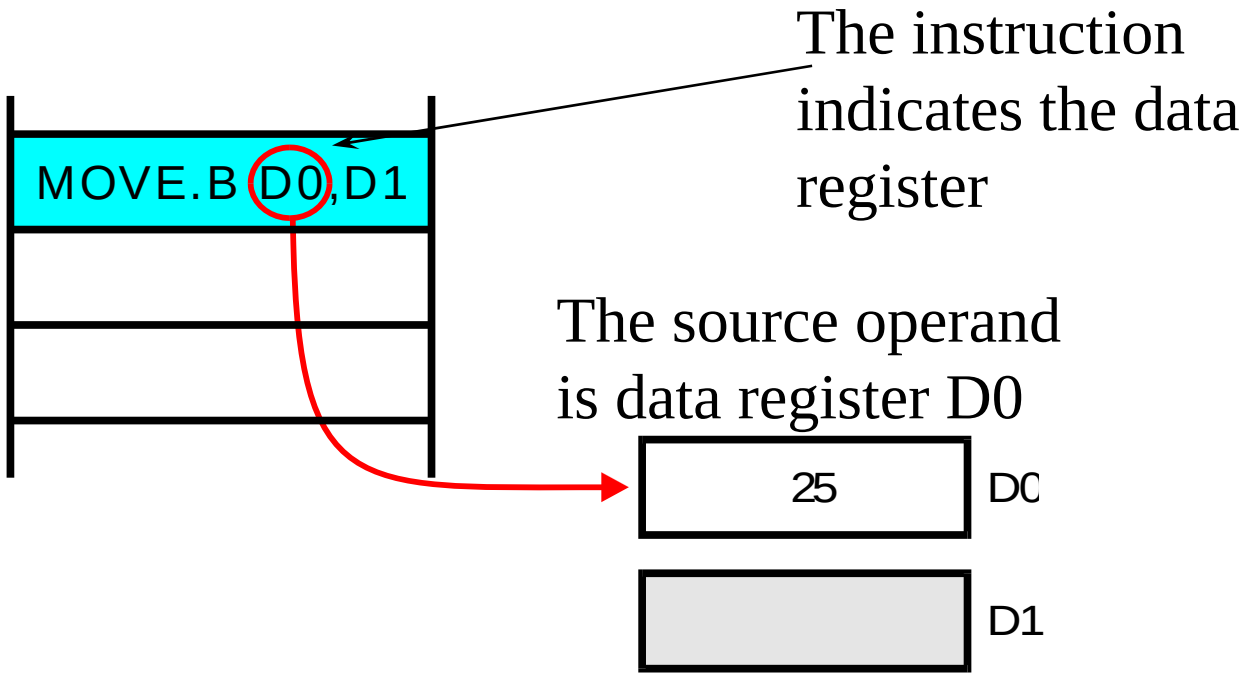
Addressing modes are concerned with how the CPU accesses the operands used by its instructions

# Register Direct Addressing

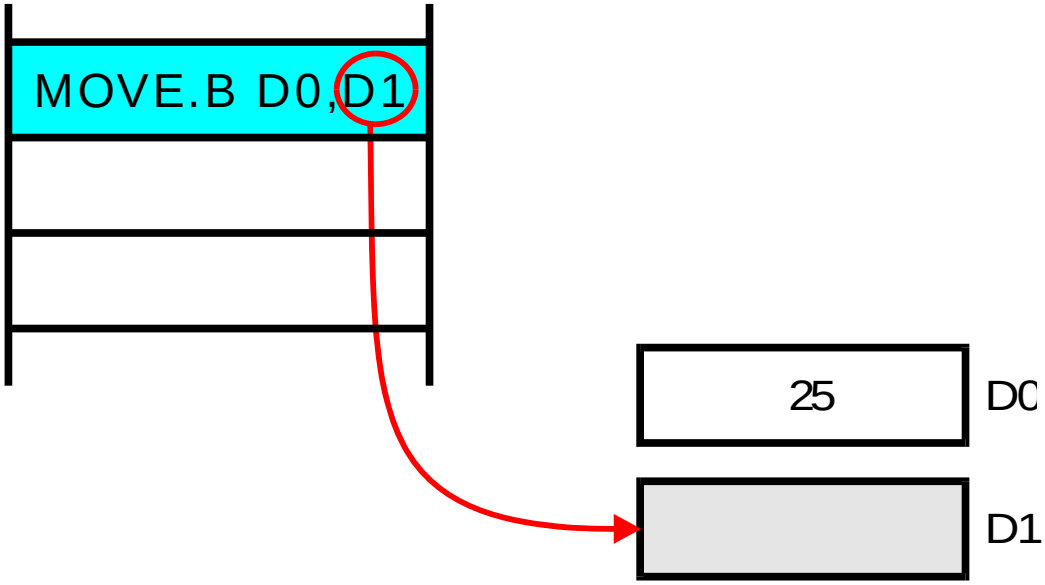
*Register direct addressing* is the simplest addressing mode in which the source or destination of an operand is a data register or an address register. The contents of the specified source register provide the source operand. Similarly, if a register is a destination operand, it is loaded with the value specified by the instruction. The following examples all use register direct addressing for source and destination operands.

```
MOVE.B D0,D3     Copy the source operand in register D0 to register D3
SUB.L  A0,D3     Subtract the source operand in register A0 from register D3
CMP.W  D2,D0     Compare the source operand in register D2 with register D0
ADD    D3,D4     Add the source operand in register D3 to register D4
```

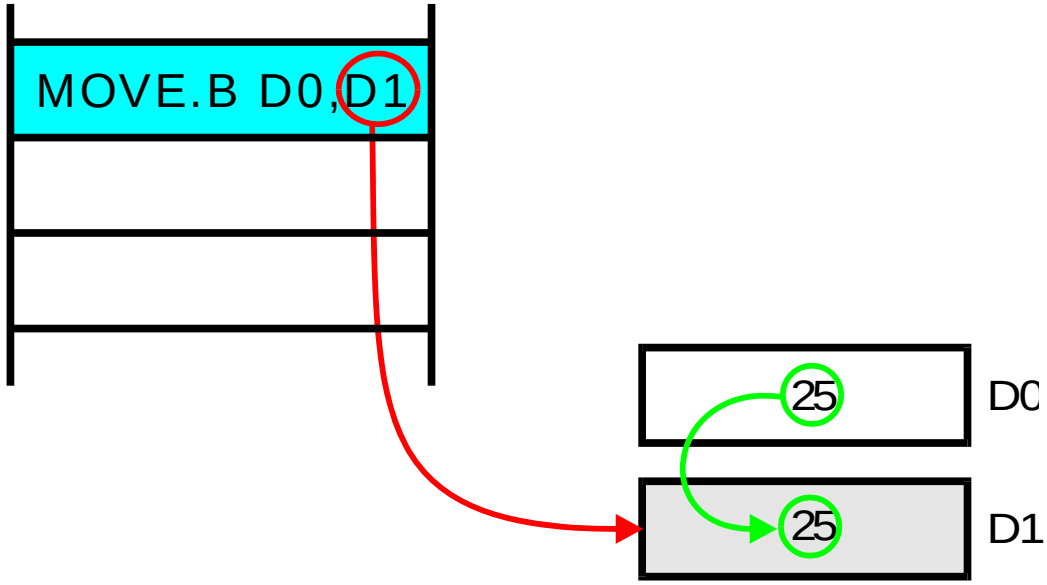
# Register Direct Addressing




The MOVE.B D0,D1 instruction uses data registers for both source and destination operands




The destination operand is data register D1




The effect of this instruction is to copy the contents of data register D0 in to data register D1

# Register Direct Addressing

Register direct addressing uses short instructions because it takes only three bits to specify one of eight data registers.

Register direct addressing is fast because the external memory does not have to be accessed.

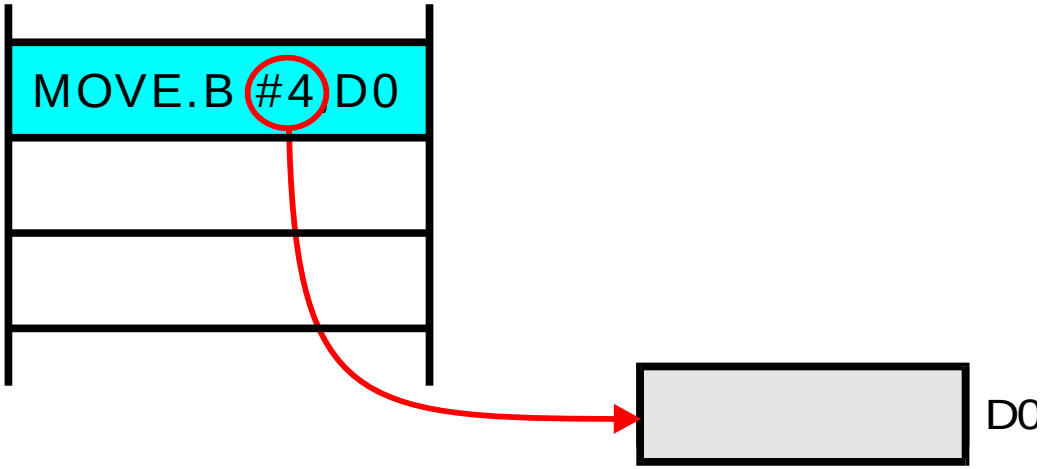
Programmers use register direct addressing to hold variables that are frequently accessed (i.e., scratchpad storage).

# Immediate Addressing

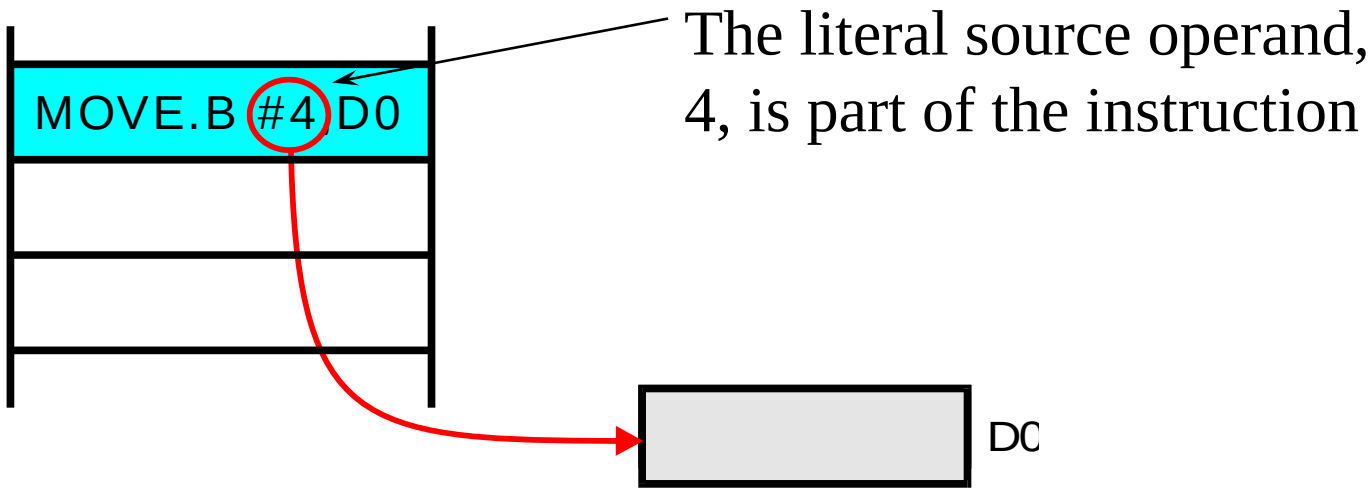
In *immediate addressing* the actual operand forms part of the instruction. An immediate operand is also called a literal operand. Immediate addressing can be used only to specify a source operand.

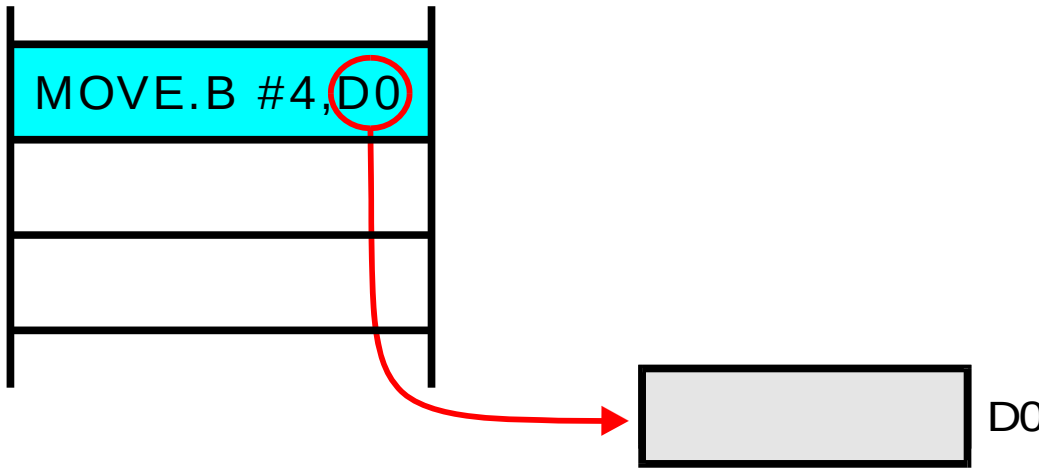
Immediate addressing is indicated by a # symbol in front of the source operand.

For example, MOVE.B #24,D0 uses the immediate source operand 24.

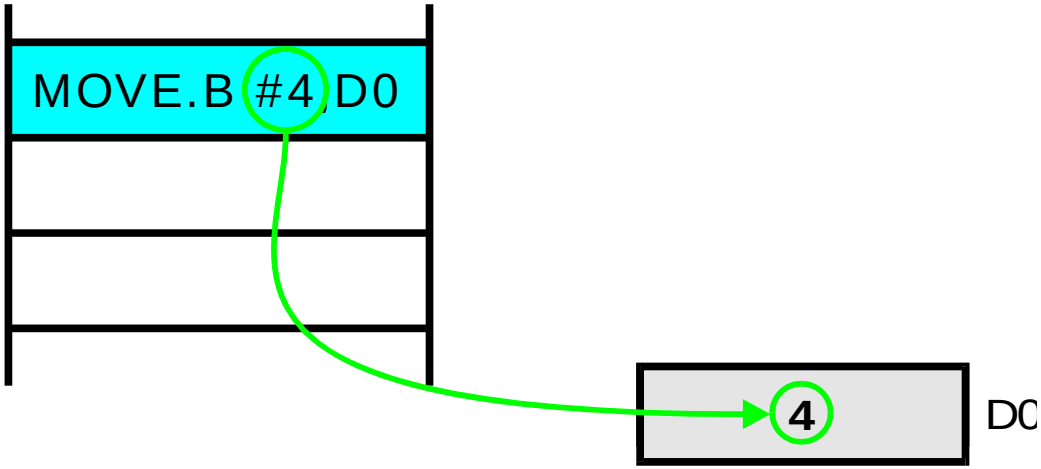



The instruction MOVE.B #4,D0
uses a literal source operand and
a register direct destination operand


MOVE.B #4,D0
The literal source operand,
4, is part of the instruction
D0




The destination operand is
a data register



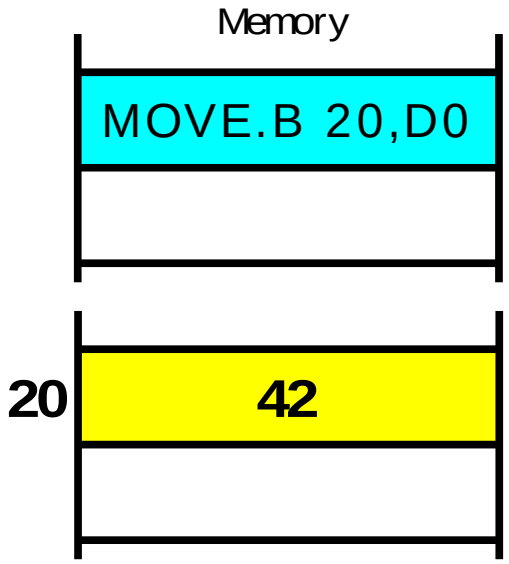

The effect of this instruction is to copy the literal value 4 to data register D0

# Direct Addressing

In *direct* or *absolute addressing*, the instruction provides the address of the operand in memory.

Direct addressing requires two memory accesses. The first is to access the instruction and the second is to access the actual operand.

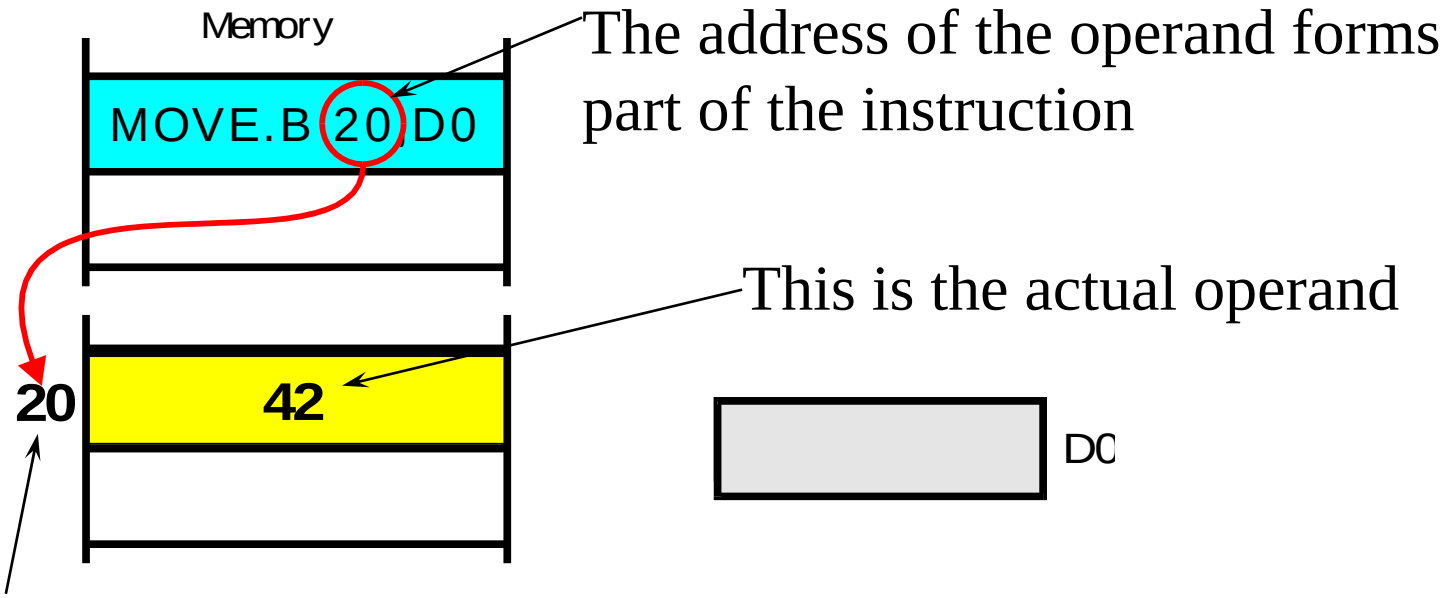
For example, CLR.B 1234 clears the contents of memory location 1234.



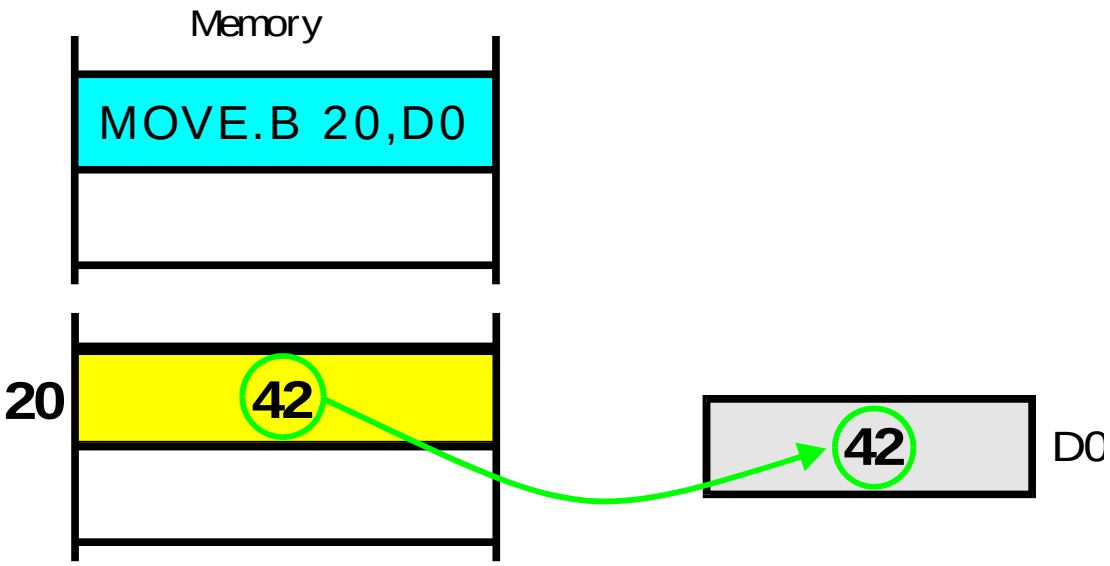

This instruction has a direct source operand

The source operand is in memory


The destination operand uses data register direct addressing




Once the CPU has read the operand address from the instruction, the CPU accesses the actual operand




The effect of MOVE.B 20,D0
is to read the contents of memory
location 20 and copy them to D0

# Summary of Fundamental Addressing Modes

Consider the high-level language example: Z = Y + 4

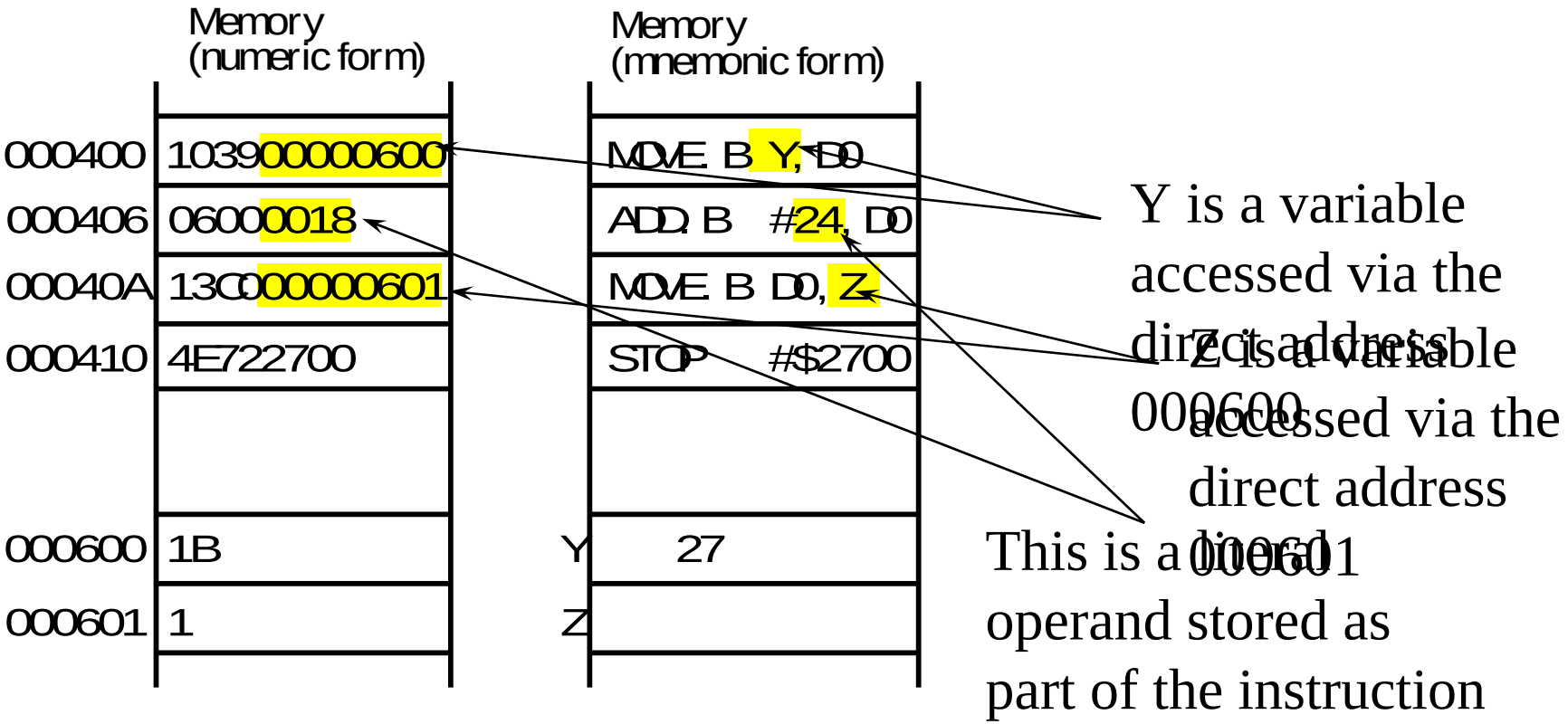
The following fragment of code implements this construct

```
    ORG     $400   Start of code
    MOVE.B  Y,D0
    ADD     #4,D0
    MOVE.B  D0,Z

    ORG     $600   Start of data area
Y   DC.B    27     Store the constant 27 in memory
Z   DS.B    1      Reserve a byte for Z
```

# The Assembled Program

```
1  00000400                ORG     $400
2  00000400 103900000600   MOVE.B  Y,D0
3  00000406 06000018       ADD.B   #24,D0
4  0000040A 13C000000601   MOVE.B  D0,Z
5  00000410 4E722700       STOP    #$2700
6                  *
7  00000600                ORG     $600
8  00000600 1B         Y:  DC.B    27
9  00000601 00000001   Z:  DS.B    1
10         00000400        END     $400
```

# Memory map of the program



# Summary

Register direct addressing is used for variables that can be held in registers

Literal (immediate) addressing is used for constants that do not change

Direct (absolute) addressing is used for variables that reside in memory

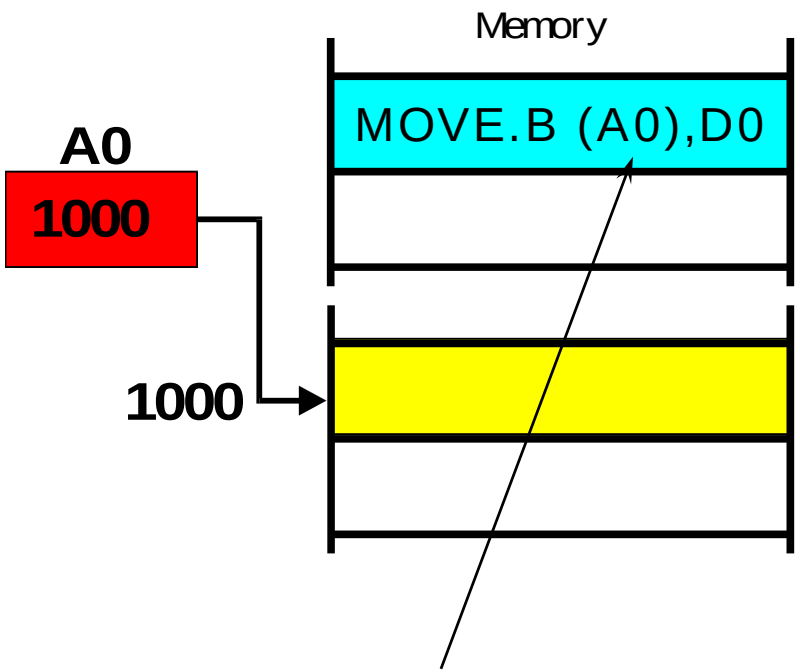
The only difference between register direct addressing and direct addressing is that the former uses registers to store operands and the latter uses memory

# Address Register Indirect Addressing

In *address register indirect addressing*, the instruction specifies one of the 68000's address registers; for example, MOVE.B (A0),D0.

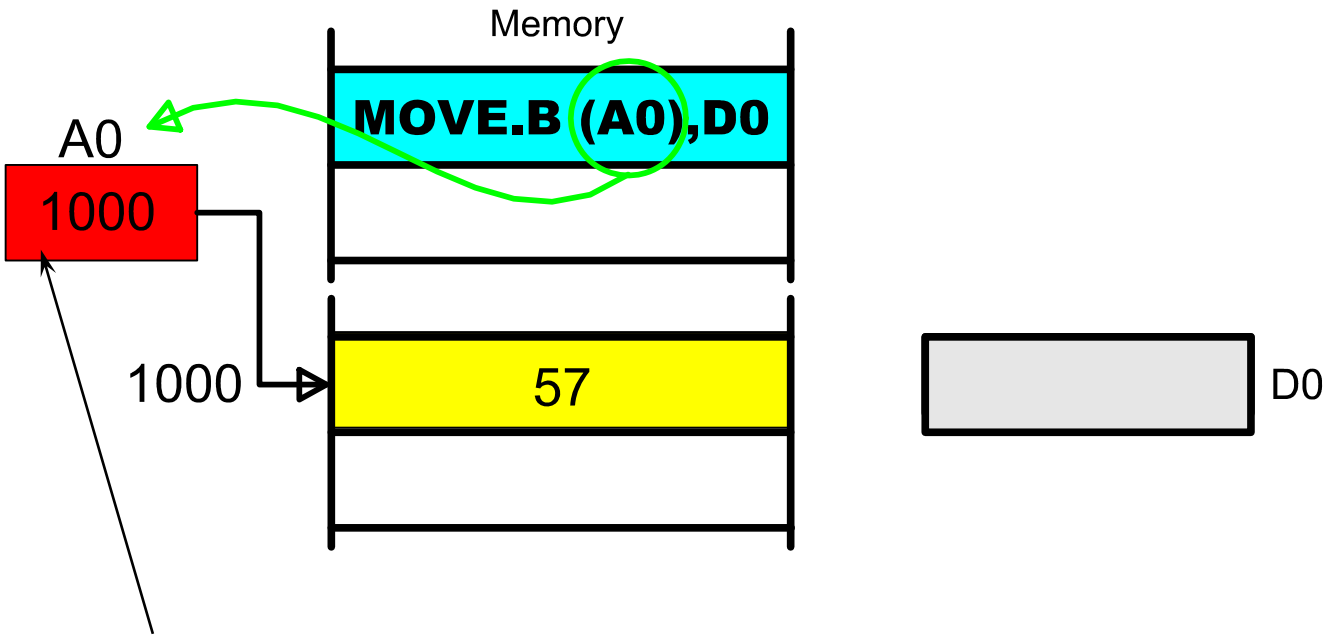
The specified address register contains the address of the operand.

The processor then accesses the operand pointed at by the address register.

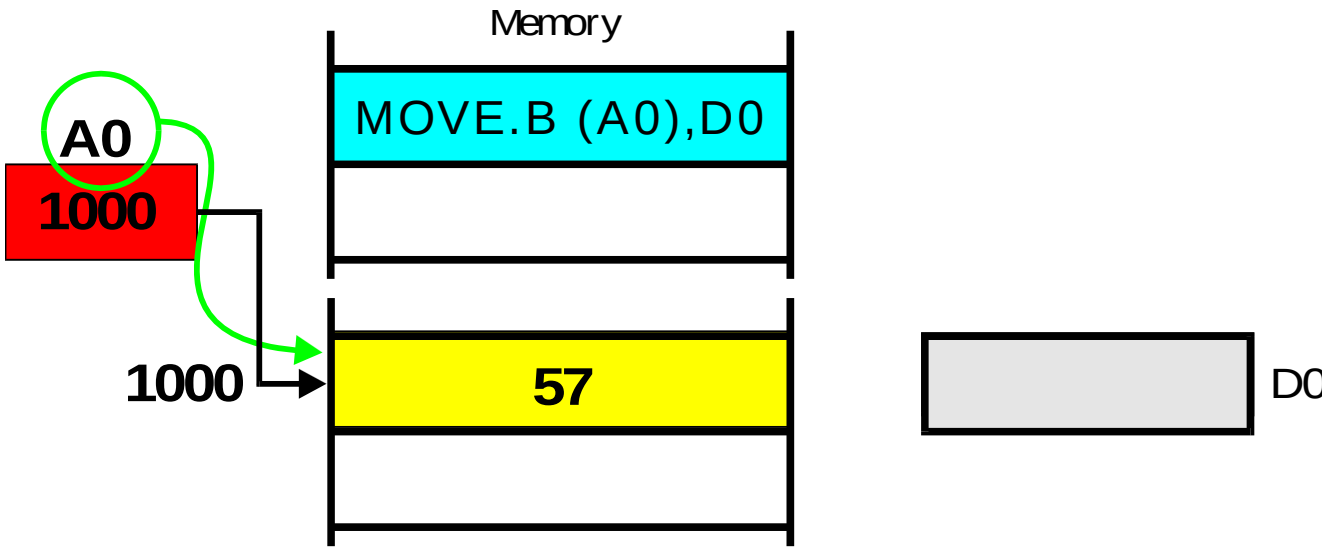



This instruction means load D0 with the contents of the location pointed at by address register A0

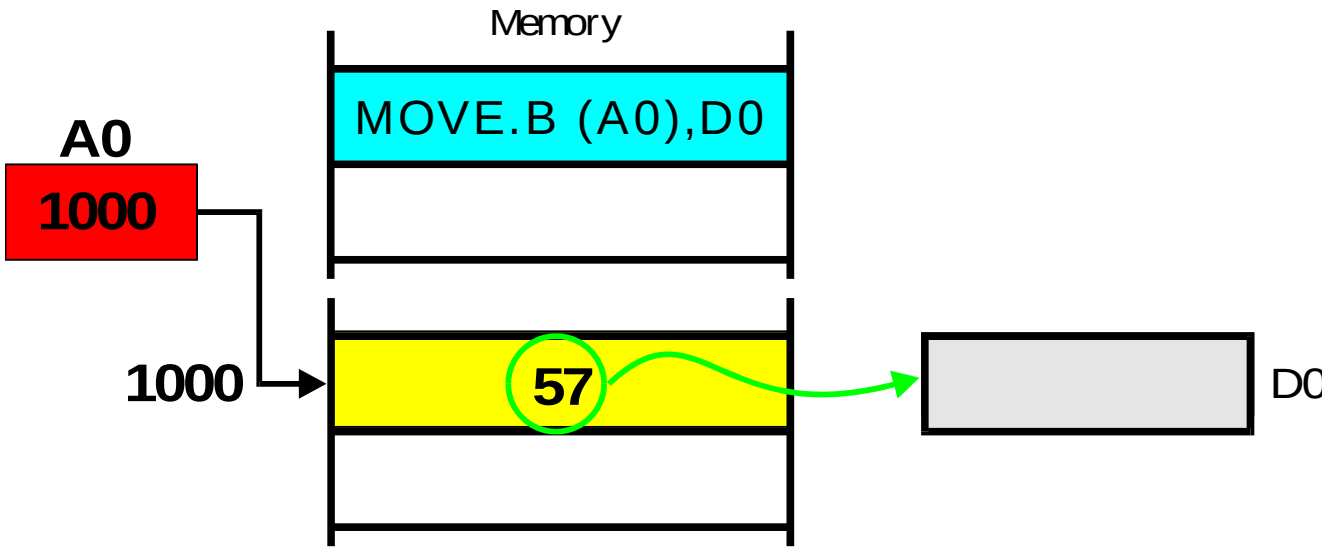

The instruction specifies the source operand as (A0).




The address register in the instruction specifies an address register that holds the address of the operand



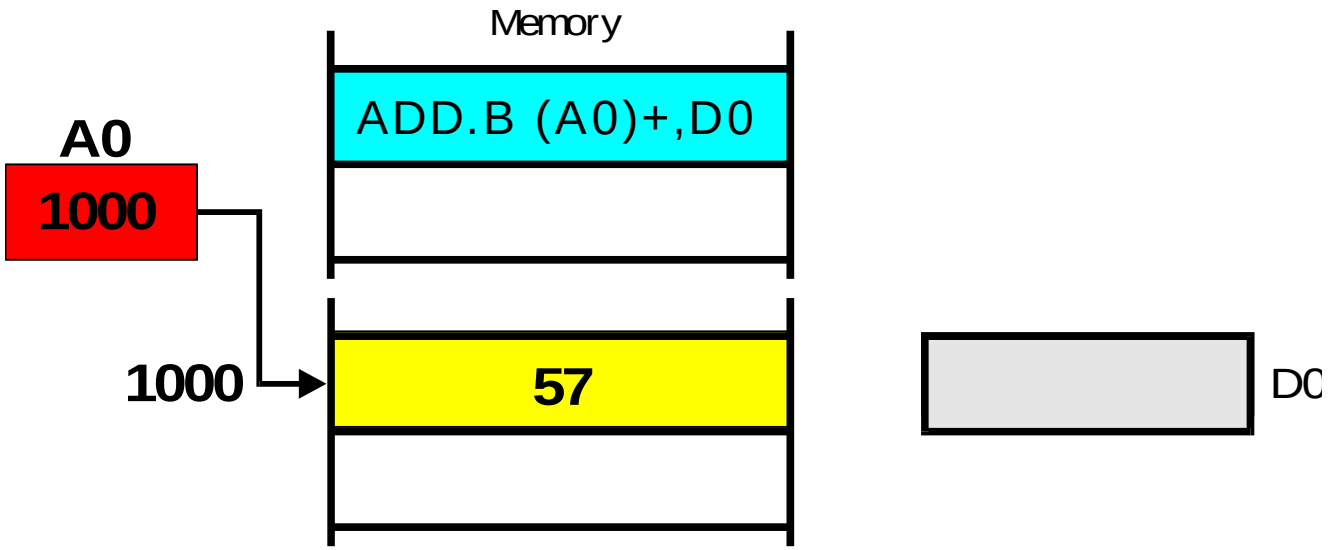

The address register is used to access the operand in memory



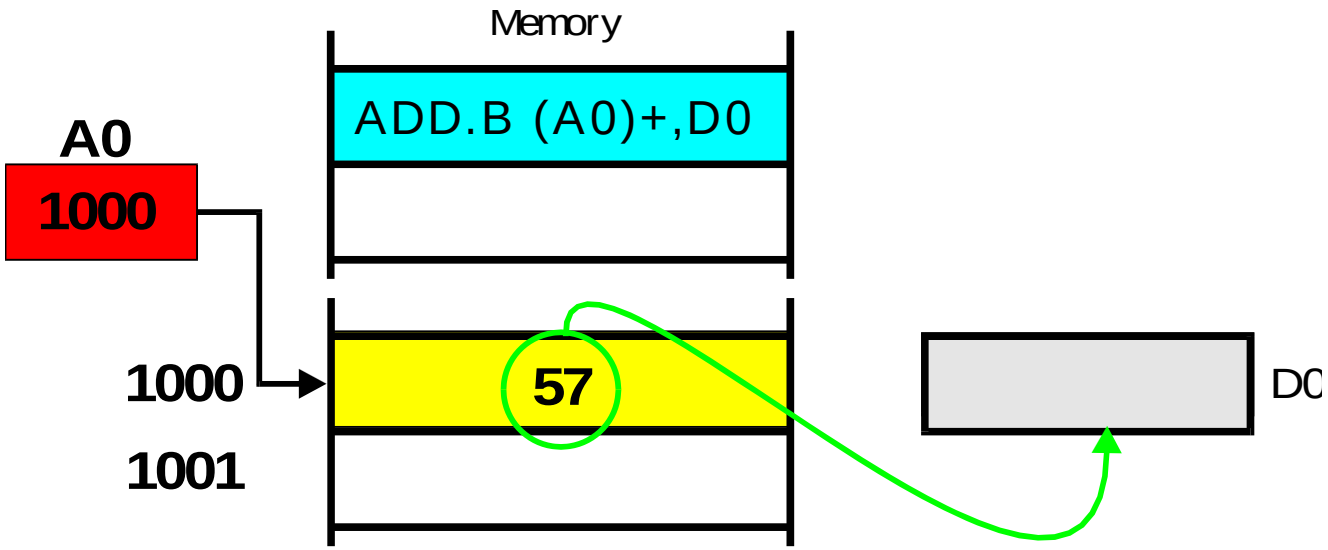

Finally, the contents of the address register pointed at by A0 are copied to the data register

# Auto-incrementing

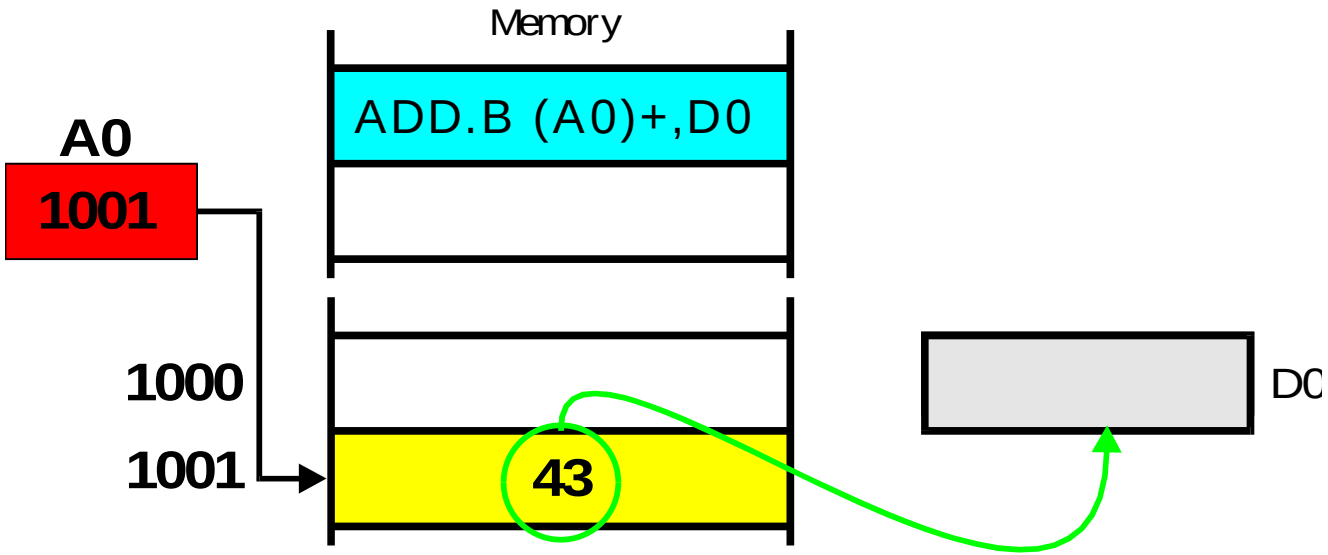
If the addressing mode is specified as (A0)+, the contents of the address register are incremented after they have been used.




The address register contains 1000
and points at location 1000




Address A0 register is used to access memory location 1000 and the contents of this location (i.e., 57) are added to D0




After the instruction has been executed, the contents of A0 are incremented to point at the next location

# Use of Address Register Indirect Addressing

The following fragment of code uses address register indirect addressing with post-incrementing to add together five numbers stored in consecutive memory locations.

```
      MOVE.B  #5,D0        Five numbers to add
      LEA     Table,A0     A0 points at the numbers
      CLR.B   D1           Clear the sum
Loop  ADD.B   (A0)+,D1     REPEAT Add number to total
      SUB.B   #1,D0
      BNE     Loop         UNTIL all numbers added
      STOP    #$2700
*
Table DC.B    1,4,2,6,5    Some dummy data
```

We are now going to trace through part of this program, instruction by instruction.

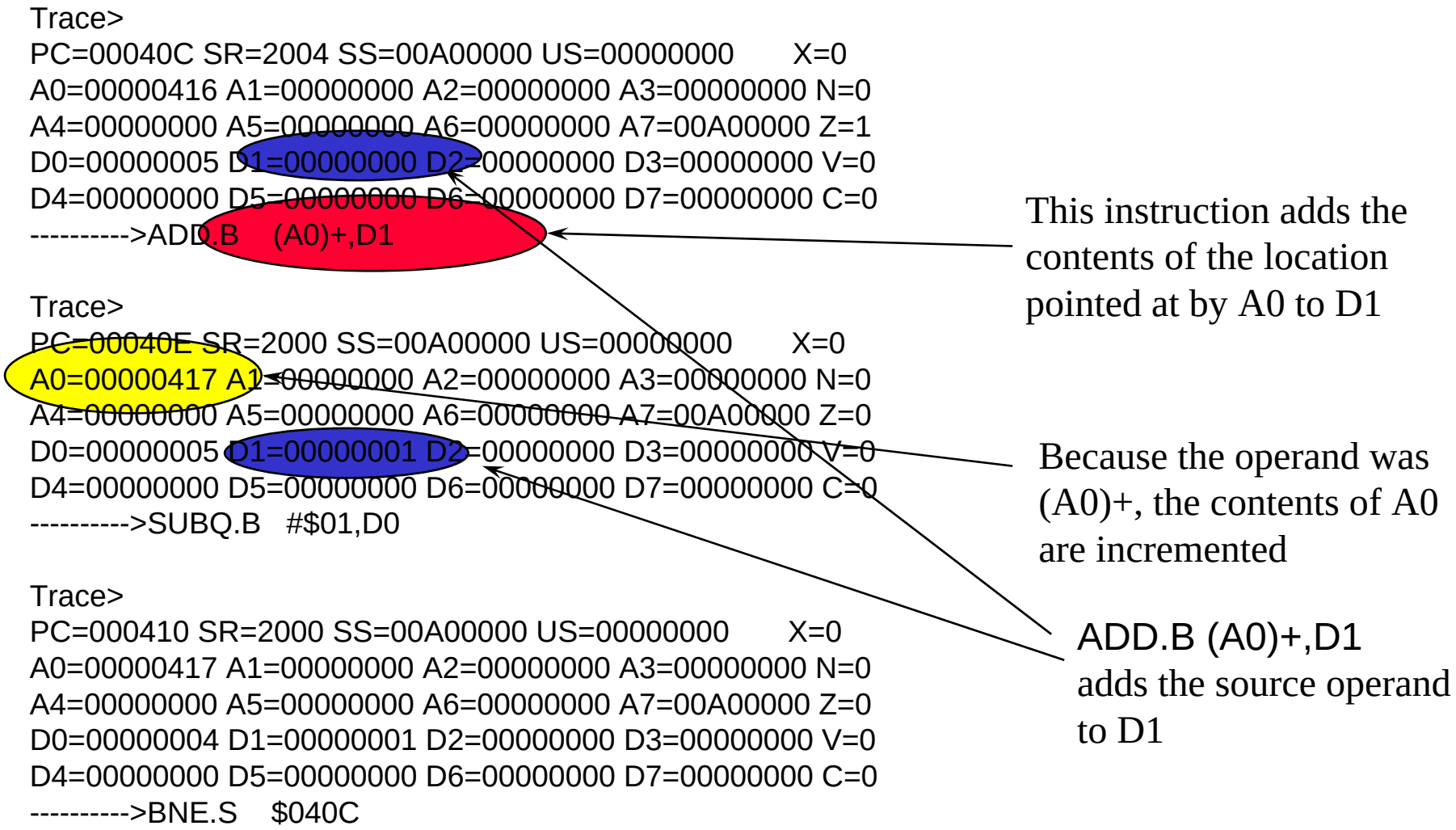
```
>DF
PC=000400 SR=2000 SS=00A00000 US=00000000       X=0
A0=00000000 A1=00000000 A2=00000000 A3=00000000 N=0
A4=00000000 A5=00000000 A6=00000000 A7=00A00000 Z=0
D0=00000000 D1=00000000 D2=00000000 D3=00000000 V=0
D4=00000000 D5=00000000 D6=00000000 D7=00000000 C=0
---------->MOVE.B   #$05,D0

>TR
PC=000404 SR=2000 SS=00A00000 US=00000000       X=0
A0=00000000 A1=00000000 A2=00000000 A3=00000000 N=0
A4=00000000 A5=00000000 A6=00000000 A7=00A00000 Z=0
D0=00000005 D1=00000000 D2=00000000 D3=00000000 V=0
D4=00000000 D5=00000000 D6=00000000 D7=00000000 C=0
---------->LEA.L    $0416,A0

Trace>
PC=00040A SR=2000 SS=00A00000 US=00000000       X=0
A0=00000416 A1=00000000 A2=00000000 A3=00000000 N=0
A4=00000000 A5=00000000 A6=00000000 A7=00A00000 Z=0
D0=00000005 D1=00000000 D2=00000000 D3=00000000 V=0
D4=00000000 D5=00000000 D6=00000000 D7=00000000 C=0
---------->CLR.B    D1
```

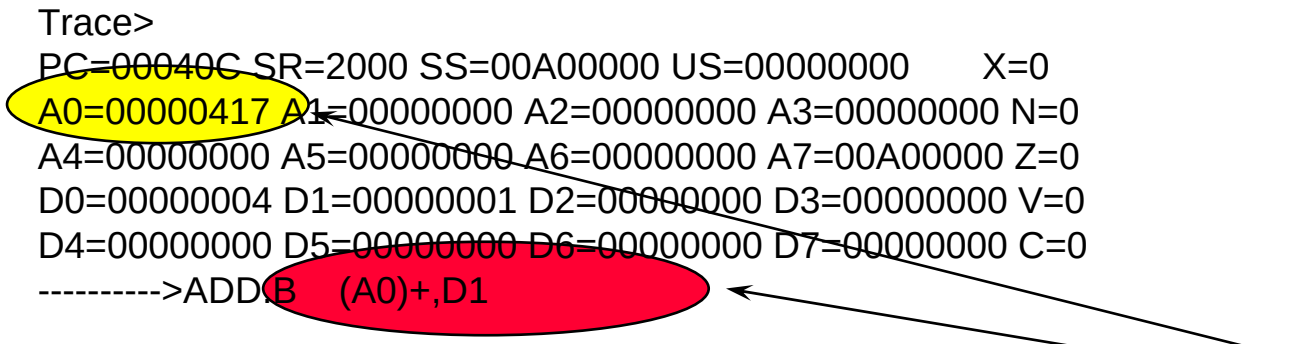
The first instruction loads D0 with the literal value 5

D0 has been loaded with 5

This instruction loads A0 with the value $0416

A0 contains $0416

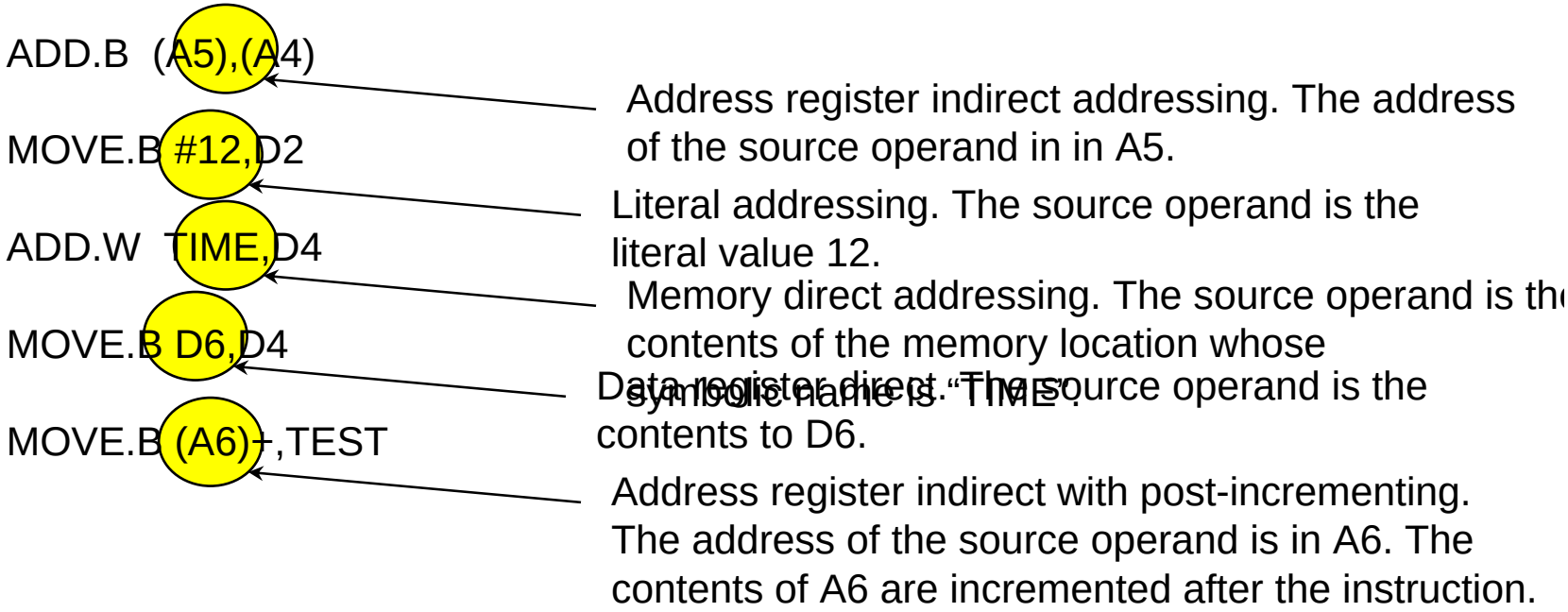

Trace>
PC=00040C SR=2004 SS=00A00000 US=00000000 X=0
A0=00000416 A1=00000000 A2=00000000 A3=00000000 N=0
A4=00000000 A5=00000000 A6=00000000 A7=00A00000 Z=1
D0=00000005 D1=00000000 D2=00000000 D3=00000000 V=0
D4=00000000 D5=00000000 D6=00000000 D7=00000000 C=0
---------->ADD.B (A0)+,D1
Trace>
PC=00040E SR=2000 SS=00A00000 US=00000000 X=0
A0=00000417 A1=00000000 A2=00000000 A3=00000000 N=0
A4=00000000 A5=00000000 A6=00000000 A7=00A00000 Z=0
D0=00000005 D1=00000001 D2=00000000 D3=00000000 V=0
D4=00000000 D5=00000000 D6=00000000 D7=00000000 C=0
---------->SUBQ.B #$01,D0
Trace>
PC=000410 SR=2000 SS=00A00000 US=00000000 X=0
A0=00000417 A1=00000000 A2=00000000 A3=00000000 N=0
A4=00000000 A5=00000000 A6=00000000 A7=00A00000 Z=0
D0=00000004 D1=00000001 D2=00000000 D3=00000000 V=0
D4=00000000 D5=00000000 D6=00000000 D7=00000000 C=0
---------->BNE.S $040C
This instruction adds the contents of the location pointed at by A0 to D1
Because the operand was (A0)+, the contents of A0 are incremented
ADD.B (A0)+,D1 adds the source operand to D1




On the next cycle the instruction ADD.B (A0)+,D1 uses A0 as a source operand and then increments the contents of A0

# Problem

Identify the source addressing mode used by each of the following instructions.



# Problem

If you were translating the following fragment of pseudocode into assembly language, what addressing modes are you most likely to use?