Modi di Indirizzamento

Calcolatori Elettronici

Formato istruzione

- CodiceOperativo
- Dimensione: quanto grandi sono gli operandi
- Operandi: valore degli operandi
- Modo di indirizzamento: se si tratta di un registro, di un accesso a memoria, di una costante, di un accesso indiretto a memoria

Problema della codifica

- Processori ortogonali: accettano tutti i modi di indirizzamento per ogni istruzione
- Scelta della codifica: …..

Formato delle istruzioni (1)

Un'istruzione si compone di:

- un codice operativo (OPCODE)
- zero, uno o più operandi

Tipi di operandi:

- operando costante
  - « esplicito (immediato)
  - « implicito (es. 0)
- operando memoria
- operando registro
  - « esplicito (es. R1)
  - « implicito (es. accumulatore)

MOVE Copy data from source to destination

Operation: `[destination] ← [source]`

Syntax: `MOVE <ea>,<e>`

Sample syntax:

```
MOVE (A5),-(A2)
MOVE -(A5),(A2)+
MOVE #$123,(A6)+
MOVE Temp1,Temp2
```

Attributes: Size = byte, word, longword

Description: Move the contents of the source to the destination location. The data is examined as it is moved and the condition codes set accordingly. Note that this is actually a *copy* command because the source is not affected by the move. The move instruction has the widest range of addressing modes of all the 68000's instructions.

Condition codes:

X	N	Z	V	C
-	*	*	0	0

ADD Add binary

Operation: `[destination] ← [source] + [destination]`

Syntax:
```
ADD <ea>,Dn
ADD Dn,<ea>
```

Attributes: Size = byte, word, longword

Description: Add the source operand to the destination operand and store the result in the destination location.

Condition codes:

X	N	Z	V	C
*	*	*	*	*

Source operand addressing modes

Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Xi)	ABS.W	ABS.L	(d,PC)	(d,PC,Xn)	imm
✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

Destination operand addressing modes

Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Xi)	ABS.W	ABS.L	(d,PC)	(d,PC,Xn)	imm
✔		✔	✔	✔	✔	✔	✔	✔			

Modi di indirizzamento

- Indicano come la CPU accede agli operandi usati dalle proprie istruzioni
- La loro funzione è quella di fornire un indirizzo effettivo (EA) per l'operando di un'istruzione
  - Es: In un'istruzione per la manipolazione di un dato, l'indirizzo effettivo è l'indirizzo del dato da manipolare
  - Es: In un'istruzione di salto, l'indirizzo effettivo è l'indirizzo dell'istruzione a cui saltare
- Sono possibili moltissimi modi di indirizzamento. Nessun processore li supporta tutti, ma il 68000 ne supporta una buona parte

Formato delle istruzioni (2)

Istruzioni a 0 operandi

Istruzioni a 1 operando

Istruzioni a 2 operandi

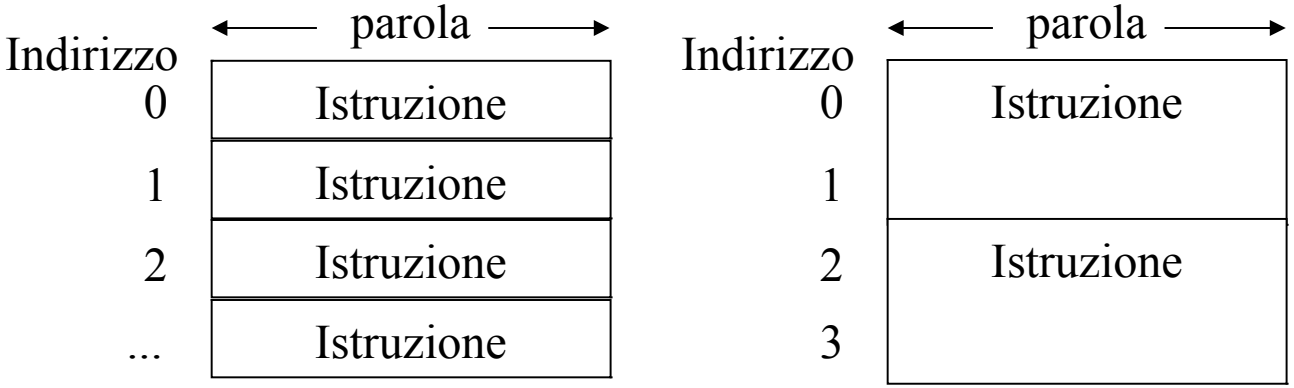
Istruzioni a 3 operandi

• Le istruzioni possono essere tutte della stessa lunghezza in bit (codifica a lunghezza fissa) oppure possono essere di lunghezze differenti (codifica a lunghezza variabile)

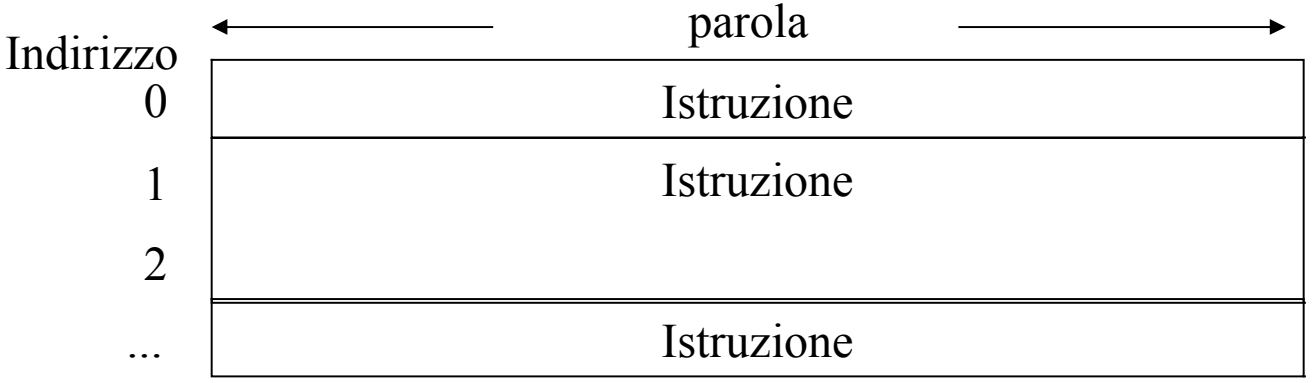
Formato delle istruzioni (3)

• Sono possibili diverse relazioni tra la lunghezza dell'istruzione e la lunghezza della parola del processore

Codifica delle istruzioni a lunghezza fissa

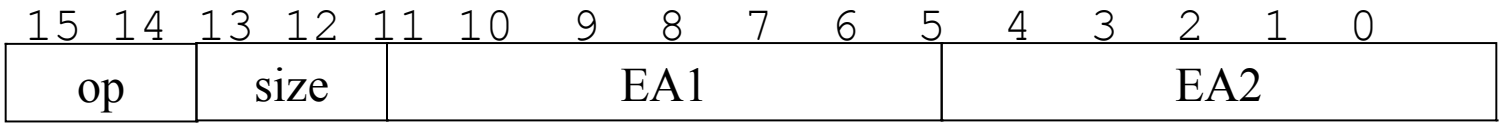


Codifica delle istruzioni a lunghezza variabile



Formato delle istruzioni: MC68000

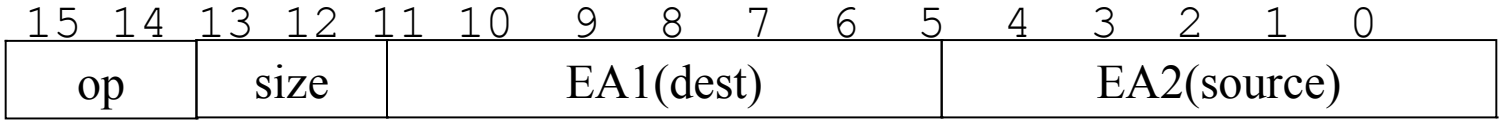
- istruzioni di lunghezza variabile da 1 a 5 parole da 16 bit
- la prima parola fornisce codice operativo, modo di indirizzamento e lunghezza dell'istruzione
- esistono differenti formati di codifica dell'opcode, di diversa lunghezza
- le parole successive contengono un operando immediato e/o un indirizzo di un operando memoria



MOVE

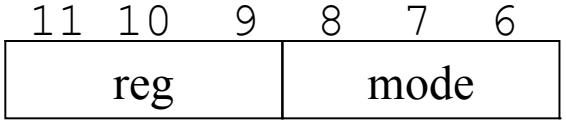
- es. istruzione MOVE src,dst
  - [src] → dst
  - src e dst possono essere operandi registro-registro, registro-memoria, memoria-registro o memoria-memoria
  - op = 00
  - size = 01 ⇒ byte size = 11 ⇒ word size = 10 ⇒ long
  - le word che cominciano con 0000 sono utilizzate per altre istruzioni
  - in definitiva 3/16 dello spazio dei codici operativi è usato da MOVE

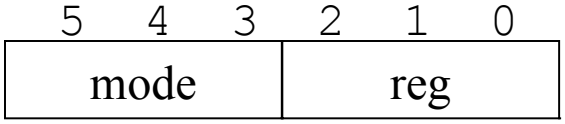
MC68000: codifica dell'istruzione MOVE



MOVE

- il campo EA di 6 bit può essere diviso in 2 sottocampi da 3 bit





alcuni modi possibili:

mode	reg	Notazione	Operando	Nome del modo di ind.	#e.w.
0	0-7	Dn	Dn	Data-register direct	0
1	0-7	An	An	Address-register direct	0
2	0-7	(An)	MEM[An]	Address-register indirect	0
7	0	addr	MEM[addr]	Absolute short	1
7	4	#data	data	Immediate	1o 2

- Gli stessi modi di indirizzamento sono possibili sia per sorgente che destinazione (ovviamente escluso l'immediato per la destinazione)

- Istruzione MOVE completamente ortogonale

MC68000: codifica dell'istruzione ADD

Istruzione ADD

- ADD.B src,Dn
- ADD.W src,Dn
- ADD.L src,Dn
- ADDA.W src,An
- ADDA.L src,An

Si utilizzano codici operativi diversi per operandi destinazione diversi.

Per src c'è un campo EA che indica uno dei modi di indirizzamento.

NOTA: Non è possibile avere la destinazione in memoria.

Esempio di istruzione non ortogonale.

Modi di Indirizzamento del 68K

- Register Direct
  - Data-register Direct
  - Address-register Direct
- Immediate (or Literal)
- Absolute
  - Short
  - Long
- Address-register Indirect
- Auto-Increment
- Auto-Decrement

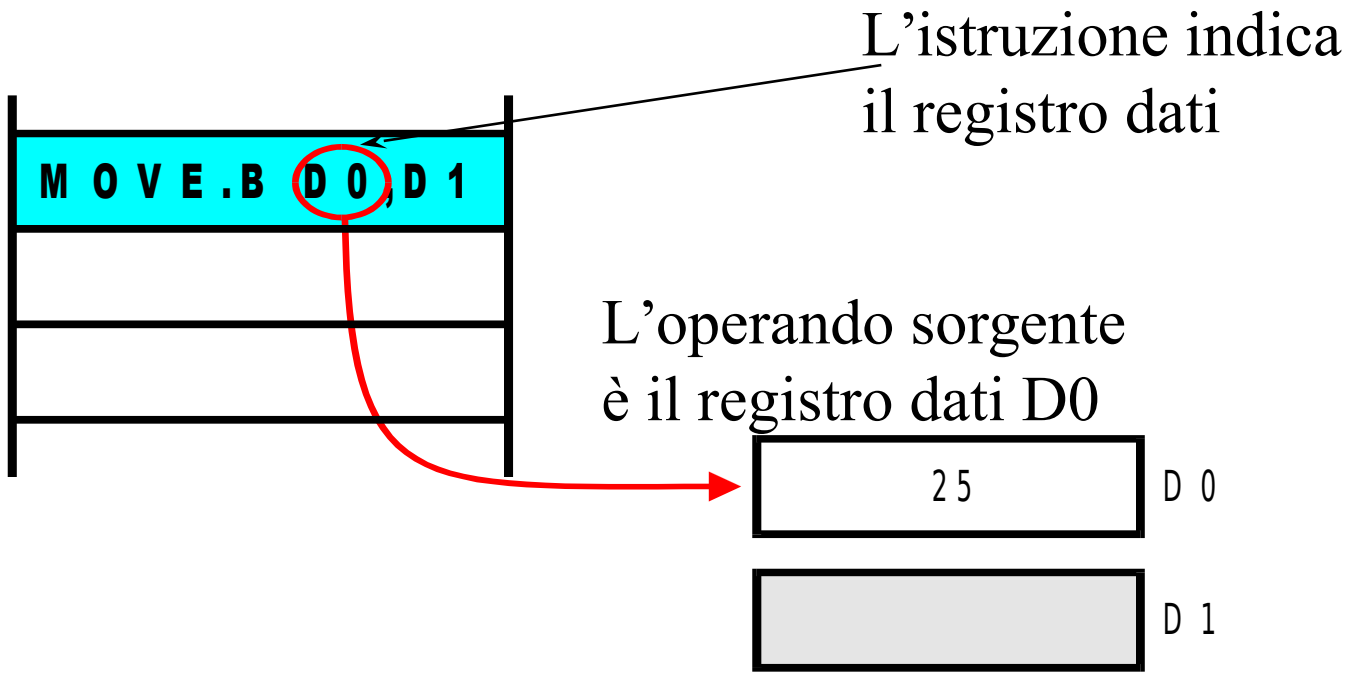
- Indexed short
- Based
- Based Indexed
  - Short
  - Long
- Relative
- Relative Indexed
  - Short
  - Long

Register Direct Addressing

- È il modo di indirizzamento più semplice
- La sorgente o la destinazione di un operando è un registro dati o un registro indirizzi
- Se il registro è un operando sorgente, il contenuto del registro specificato fornisce l'operando sorgente
- Se il registro è un operando destinazione, esso viene caricato con il valore specificato dall'istruzione

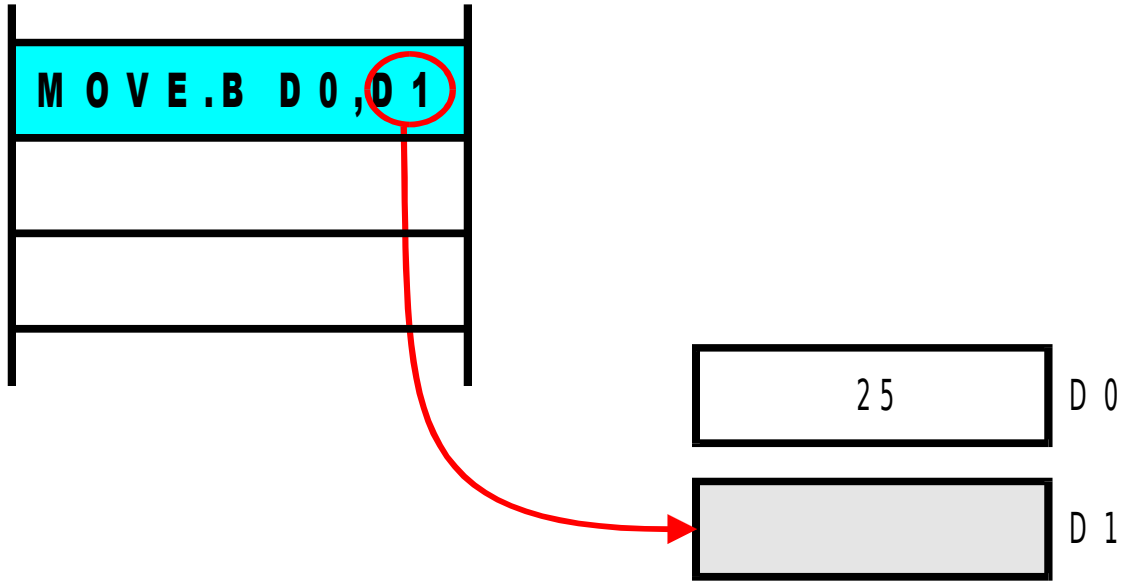
MOVE.B D0,D3	Copia l'operando sorgente in D0 nel registro D3
SUB.L A0,D3	Sottrae l'operando sorgente nel registro A0 dal registro D3
CMP.W D2,D0	Confronta l'operando sorgente nel registro D2 con il registro D0
ADD D3,D4	Somma l'operando sorgente nel registro D3 al registro D4

Register Direct Addressing – Funzionamento



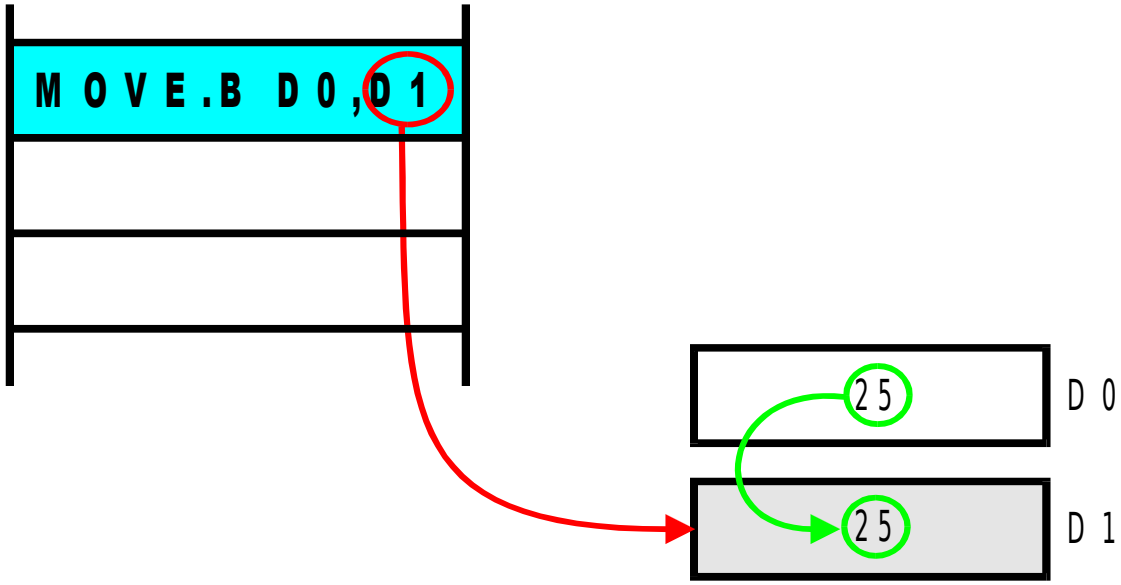
L'istruzione MOVE.B D0,D1 usa registri
dati sia per l'operando sorgente che per quello destinazione

Register Direct Addressing – Funzionamento



L'operando destinazione
è il registro dati D1

Register Direct Addressing – Funzionamento



L’effetto di questa istruzione è quello di copiare il contenuto del registro dato D0 nel registro dati D1

Register Direct Addressing - Caratteristiche

- È veloce, perché non c'è bisogno di accedere alla memoria esterna
- Fa uso di istruzioni corte, perché usa soltanto tre bit per specificare uno degli otto registri dati
  - Mode = 0, reg = 0-7 per Dn
  - Mode = 1, reg = 0-7 per An
- I programmatori lo usano per memorizzare variabili che sono usate di frequente (scratchpad storage)

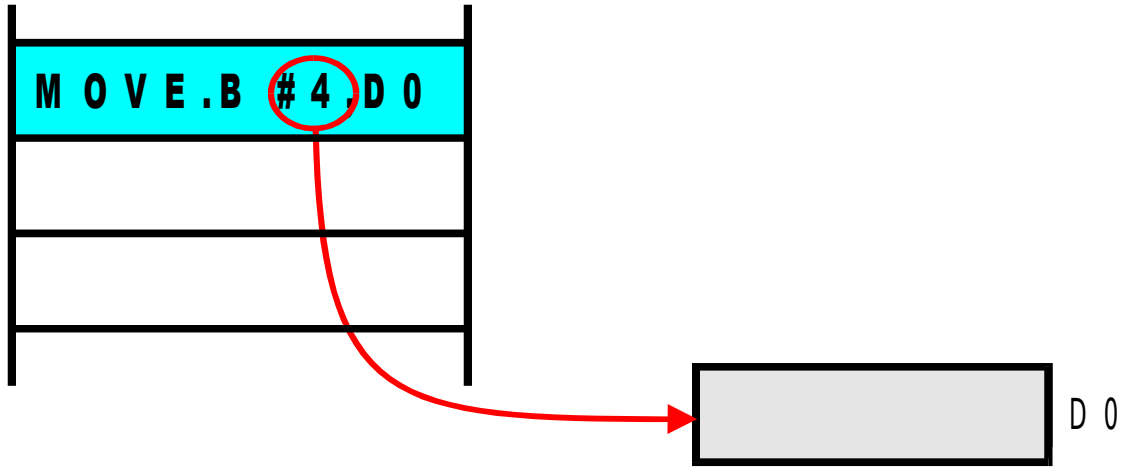
Immediate Addressing

- L'operando effettivo costituisce parte dell'istruzione
- Può essere usato unicamente per specificare un operando sorgente
- È indicato da un simbolo # davanti all'operando sorgente
- Un operando immediato è anche chiamato literal

Esempio:

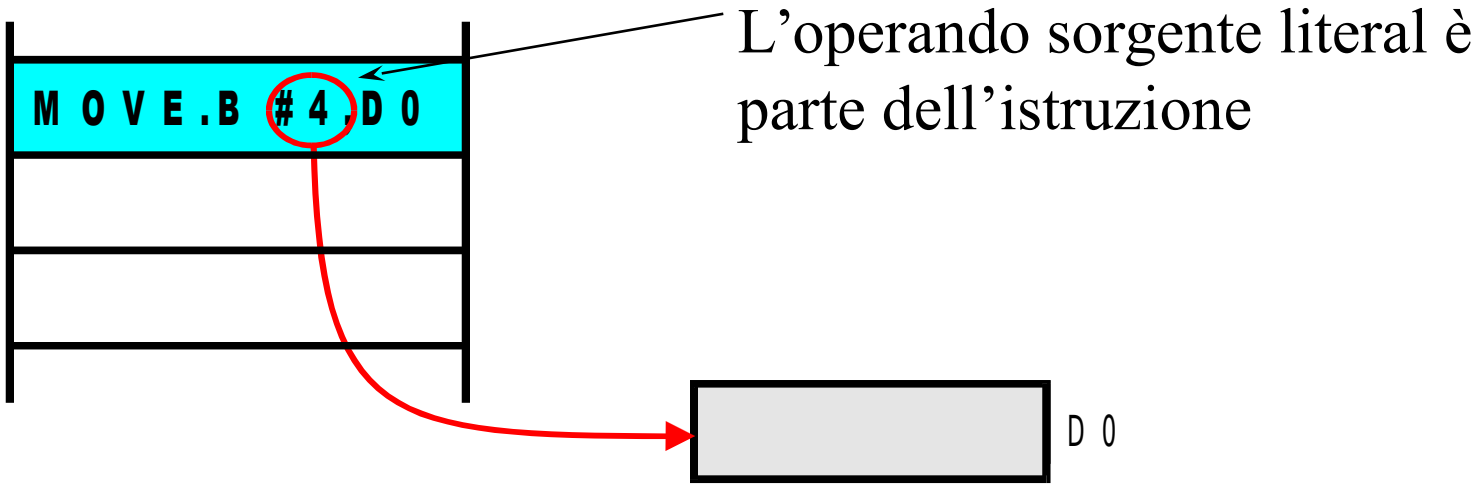
MOVE.B #4,D0 Usa l'operando sorgente immediato 4

ImmediateAddressing - Funzionamento

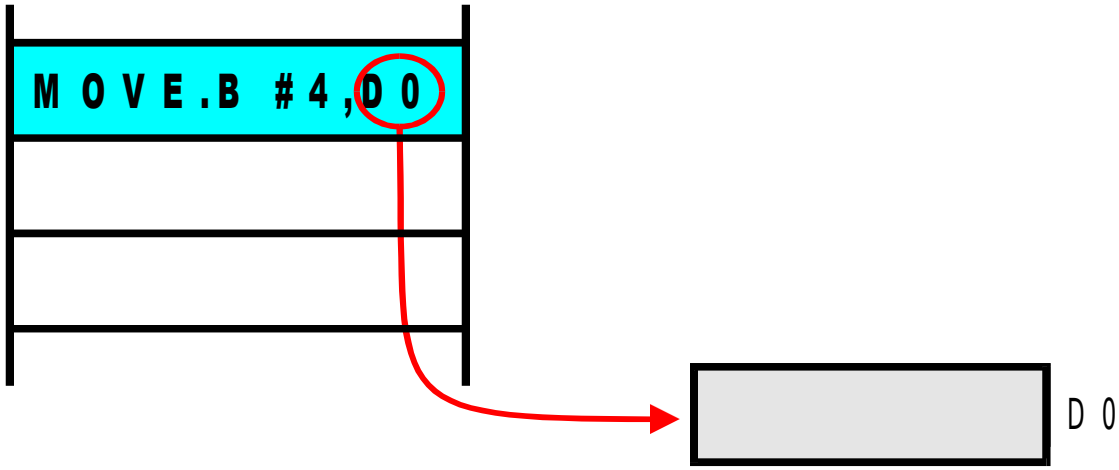


L'istruzione MOVE.B #4,D0 usa un operando sorgente literal ed un operando destinazione register direct

ImmediateAddressing - Funzionamento

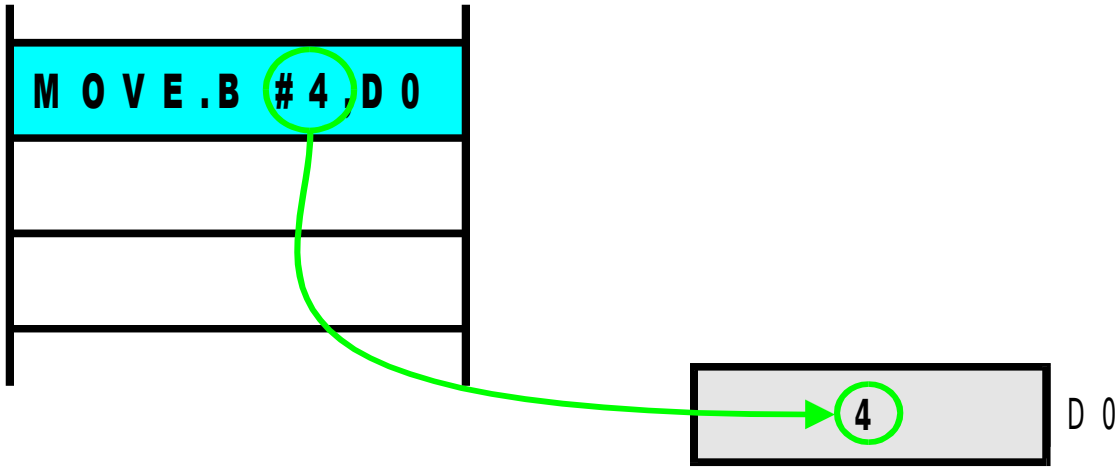


ImmediateAddressing - Funzionamento



L’operando destinazione è un registro dati

ImmediateAddressing - Funzionamento

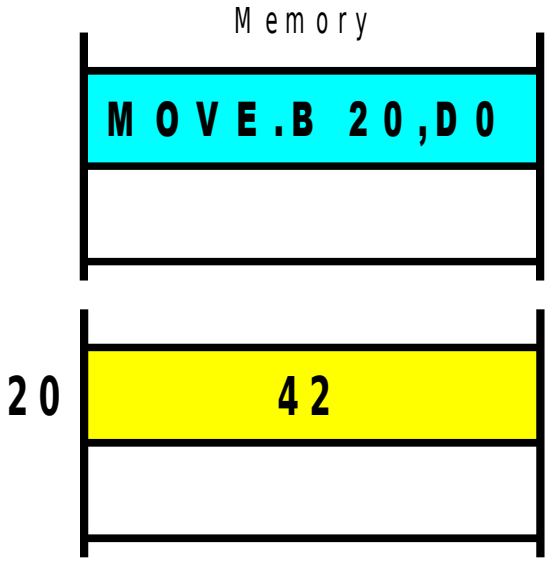


L’effetto di questa istruzione è quello di copiare il valore del literal 4 nel registro dati D0

Absolute Addressing (o Direct Addressing)

- È il modo più semplice per specificare un indirizzo di memoria completo
- L'istruzione fornisce l'indirizzo dell'operando in memoria
- Richiede due accessi in memoria:
  - Il primo è per prelevare l'istruzione
  - Il secondo è per accedere all'operando effettivo
- Esempio:
  - CLR.B 1234 azzera il contenuto della locazione di memoria 1234.

Absolute Addressing - Funzionamento

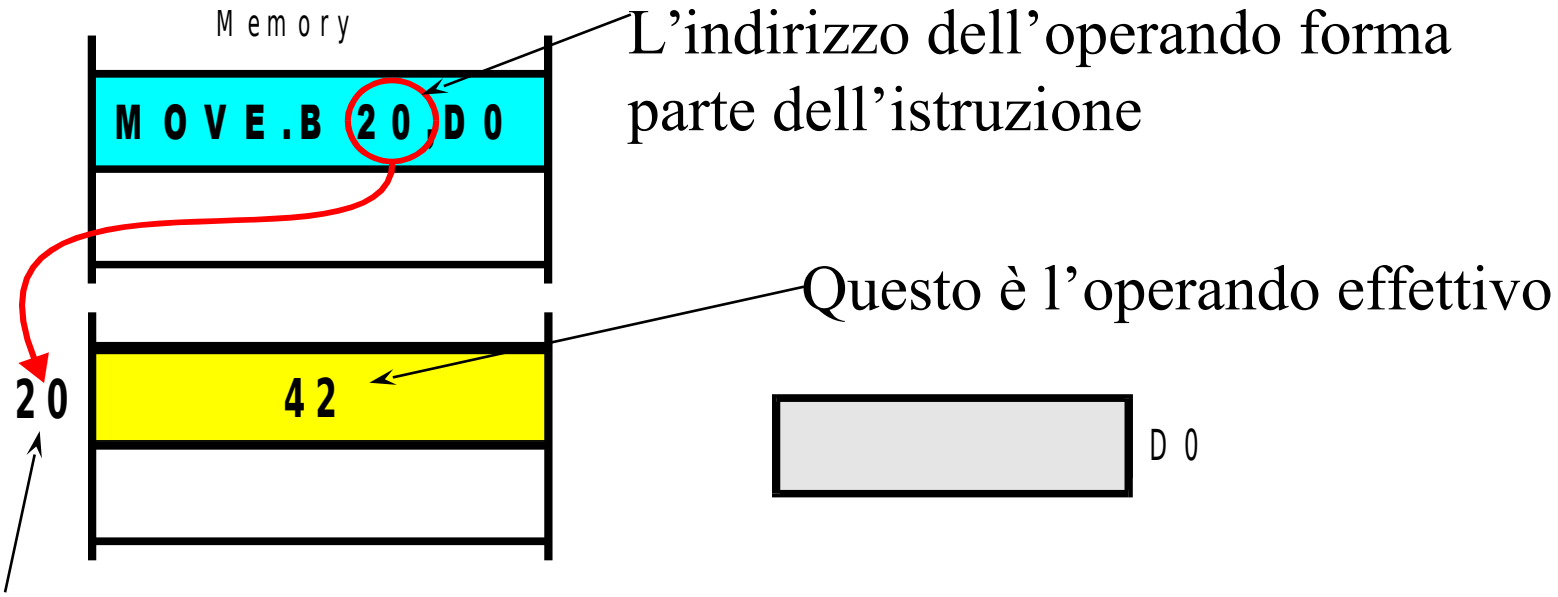


Questa istruzione ha un operando absolute

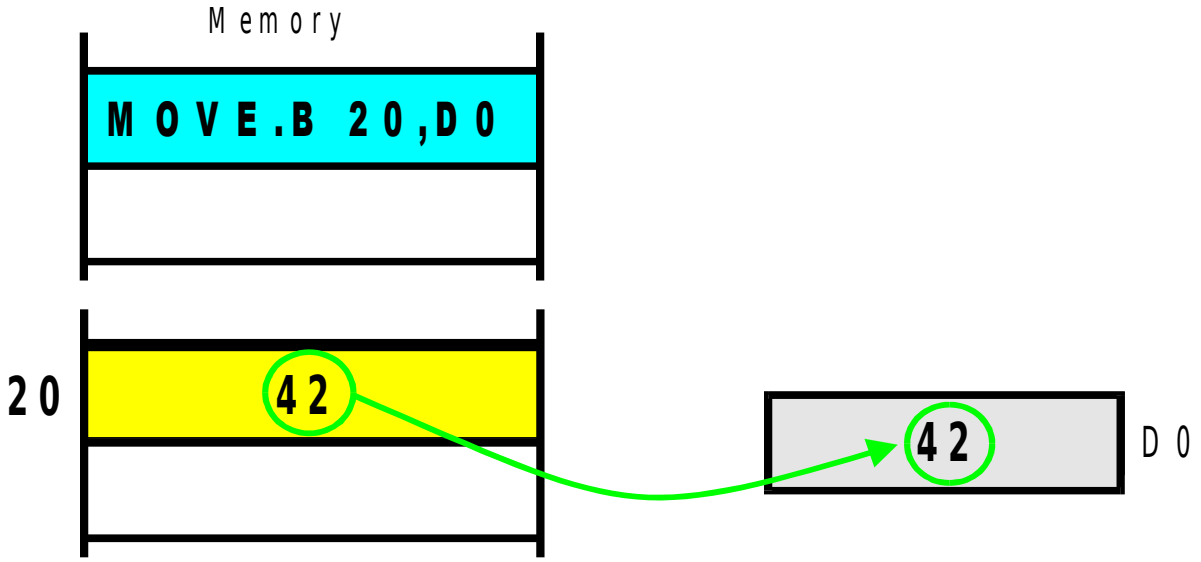
L’operando sorgente è in memoria

L’operando destinazione usa il direct addressing per un registro dati

Absolute Addressing - Funzionamento



Absolute Addressing - Funzionamento



L'effetto di MOVE.B 20,D0
è quello di leggere il contenuto della locazione
di di memoria 20 e copiarlo nel registro D0

Esempio modi fondamentali

Consideriamo questo statement in linguaggio di alto livello:
Z = Y + 24

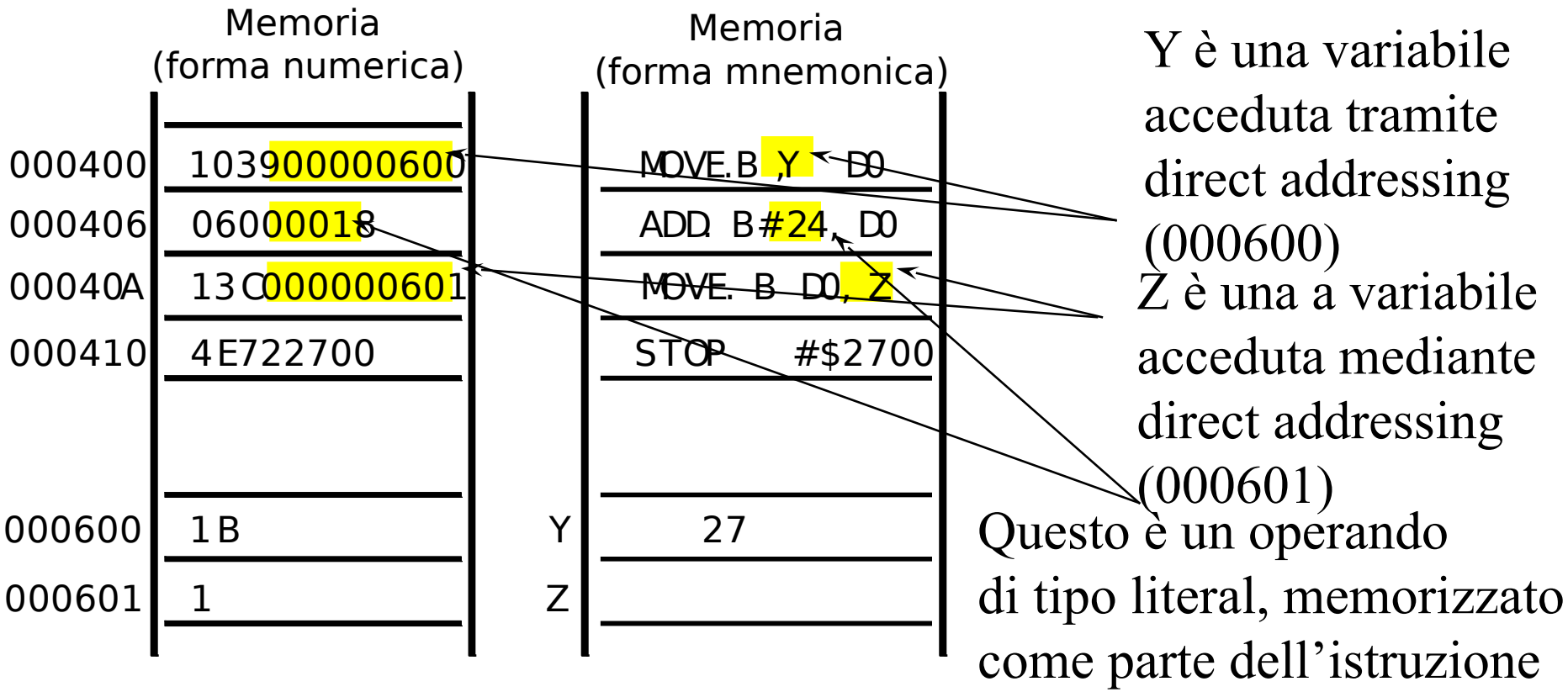
Il seguente frammento di codice implementa questo costrutto

```
        ORG     $400      Inizio del codice
        MOVE.B  Y,D0
        ADD     #24,D0
        MOVE.B  D0,Z

        ORG     $600      Inizio dell'area dati
Y       DC.B    27        Memorizza la costante 27 in memoria
Z       DS.B     1        Riserva un byte per Z
```

Il Programma Assemblato

```
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
```

Mappa della Memoria del programma



Riepilogo modi fondamentali

Register direct addressing - È usato per variabili che possono essere mantenute in registri di memoria

Literal (immediate) addressing - È usato per costanti che non cambiano

Direct (absolute) addressing - È usato per variabili che risiedono in memoria

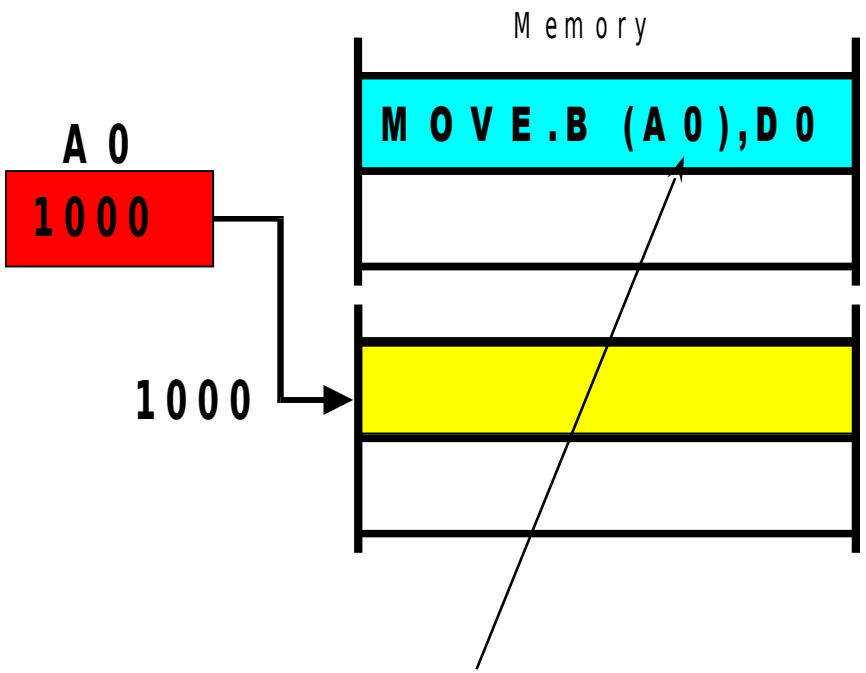
L'unica differenza tra register direct addressing e direct addressing è che il primo usa registri per memorizzare gli operandi e il secondo usa la memoria

Address Register Indirect Addressing

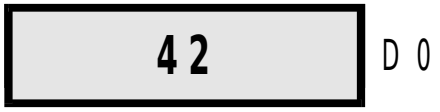
- L'istruzione specifica uno dei registri indirizzo
- Il registro indirizzo specificato contiene l'indirizzo effettivo dell'operando
- Il processore accede all'operando puntato dal registro indirizzo

- Esempio:
  - MOVE.B (A0),D0

Address Register Indirect - Funzionamento

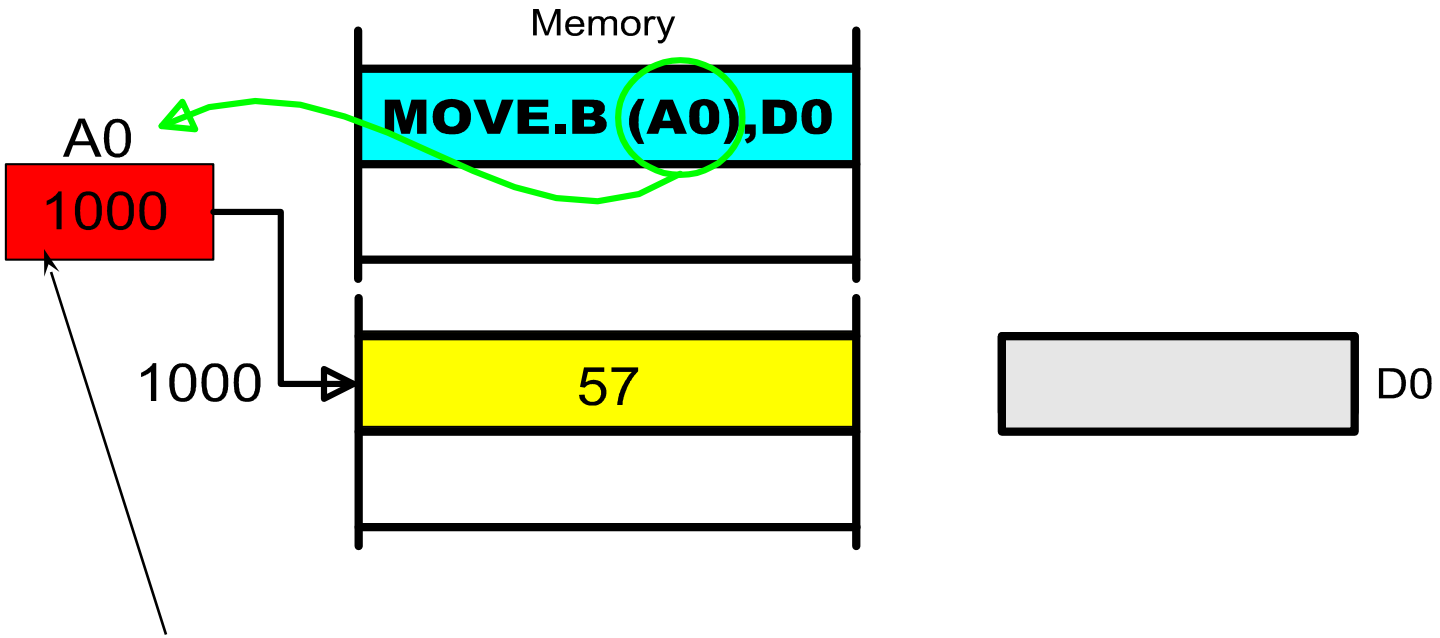


Questa istruzione significa: carica D0 con il contenuto della locazione puntata dal registro indirizzo A0



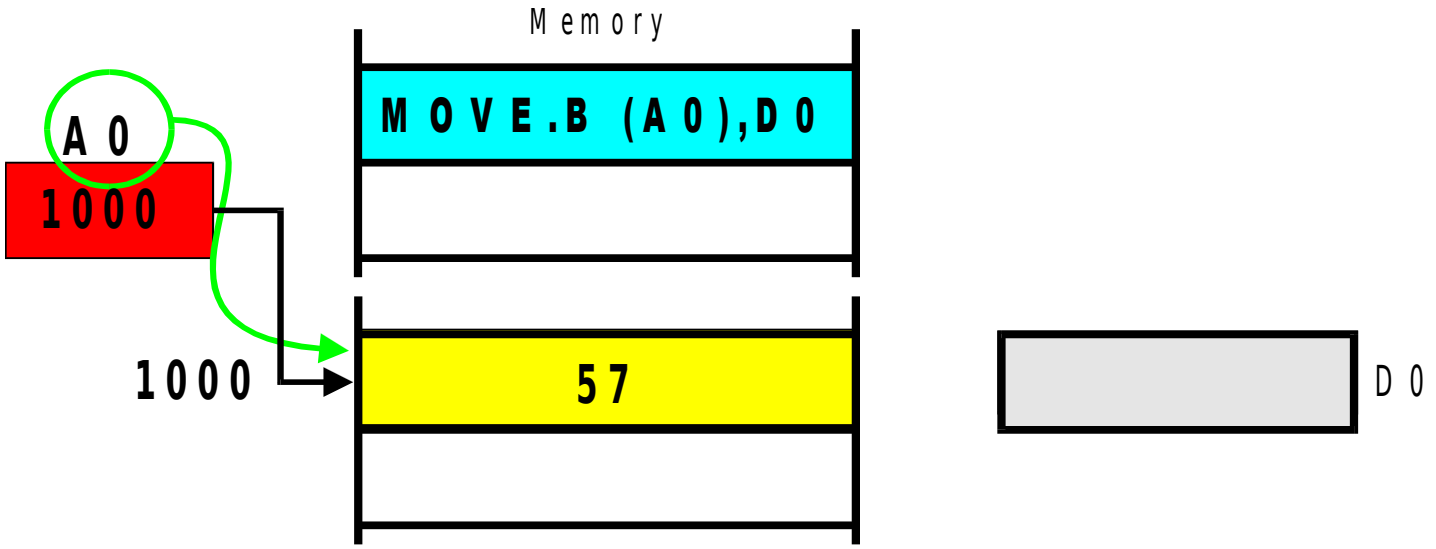
L'istruzione specifica l'operando sorgente come (A0)

Address Register Indirect - Funzionamento



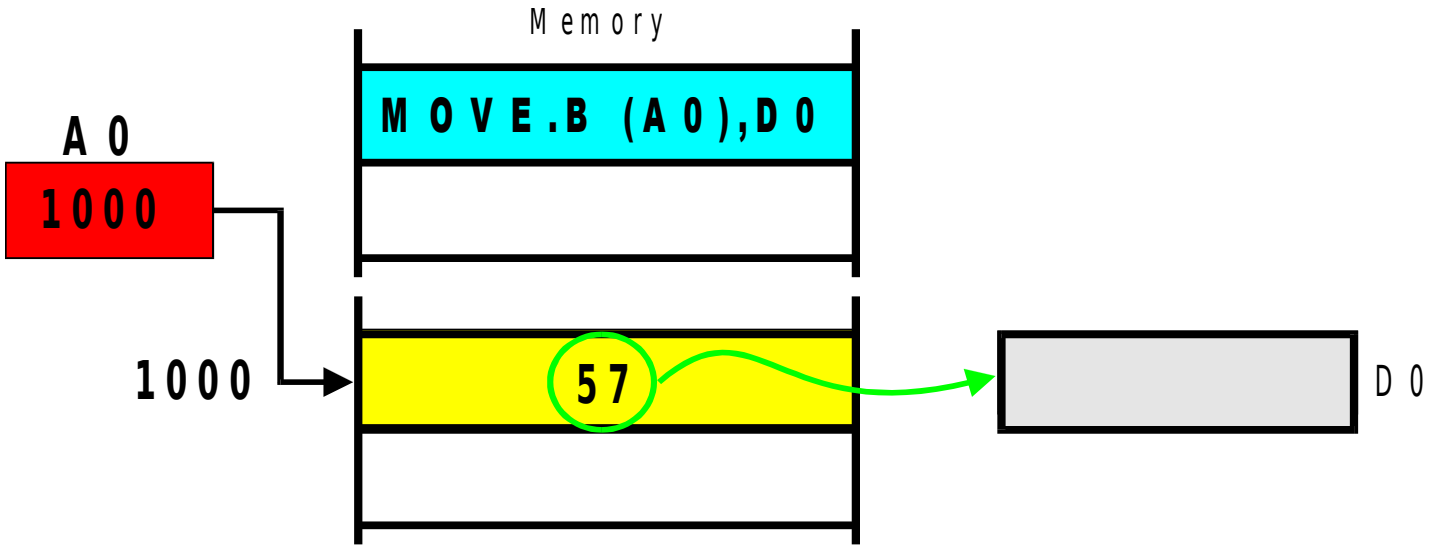
Il registro indirizzo nell'istruzione
specifica un registro indirizzo che contiene l'indirizzo dell'operando

Address Register Indirect - Funzionamento



Il registro indirizzo è usato per accedere all'operando in memoria

Address Register Indirect - Funzionamento



Alla fine, il contenuto della locazione puntata da A0 viene copiato nel registro dati

Auto-increment

- L’istruzione specifica uno dei registri indirizzo
- Se il modo di indirizzamento è specificato come (An)+, il contenuto del registro indirizzo è incrementato dopo l’uso di una quantità pari alla dimensione dell’operando

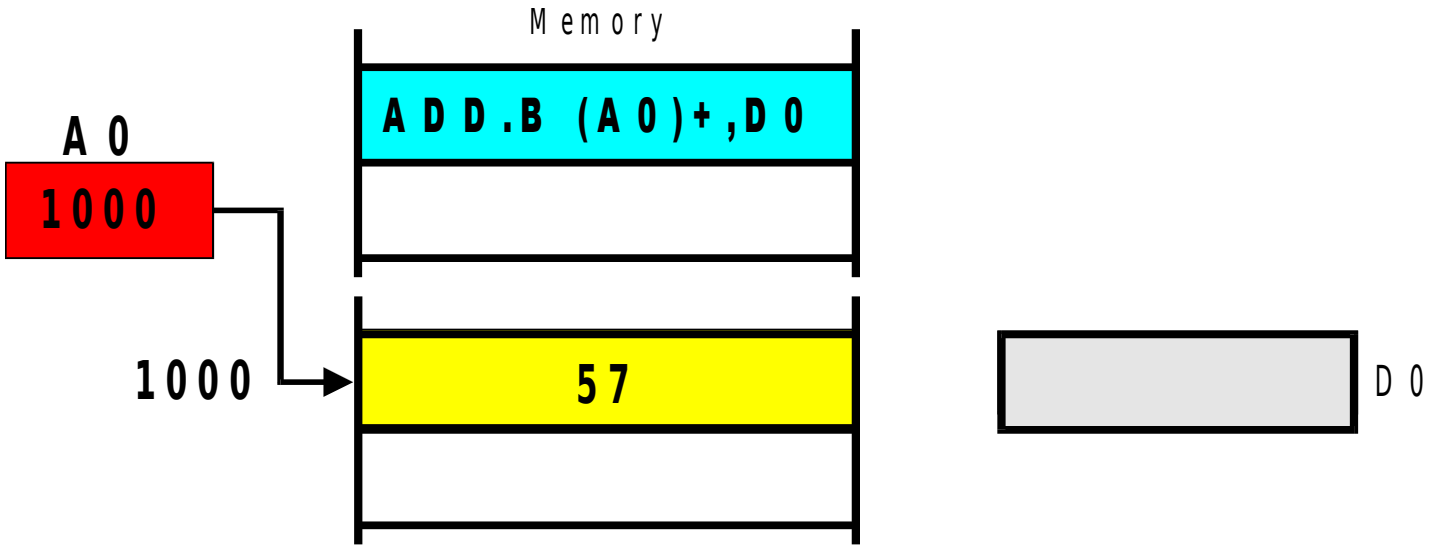
- Esempio:
  - MOVE.W (A0)+, D0 Esegue pop in D0 dallo stack di A0

Auto-decrement

- L’istruzione specifica uno dei registri indirizzo
- Se il modo di indirizzamento è specificato come -(An), il contenuto del registro indirizzo è decrementato prima dell’uso di una quantità pari alla dimensione dell’operando

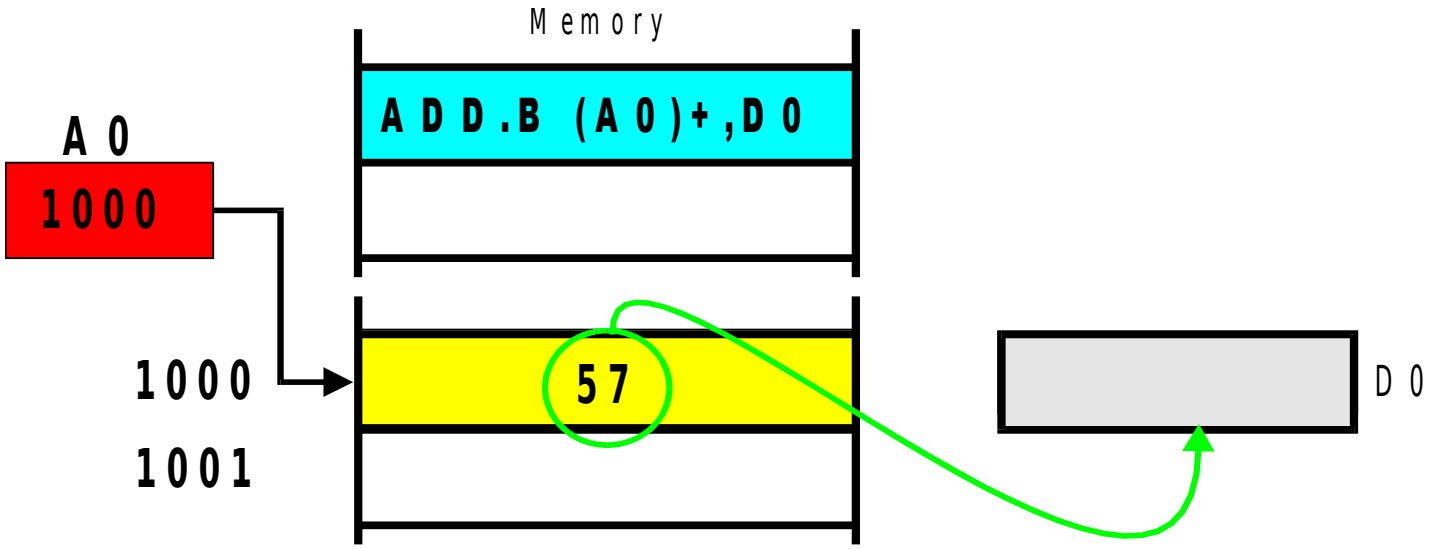
- Esempio:
  - MOVE.W D0,-(A0) Esegue push di D0 sullo stack di A0

Auto-increment - Funzionamento



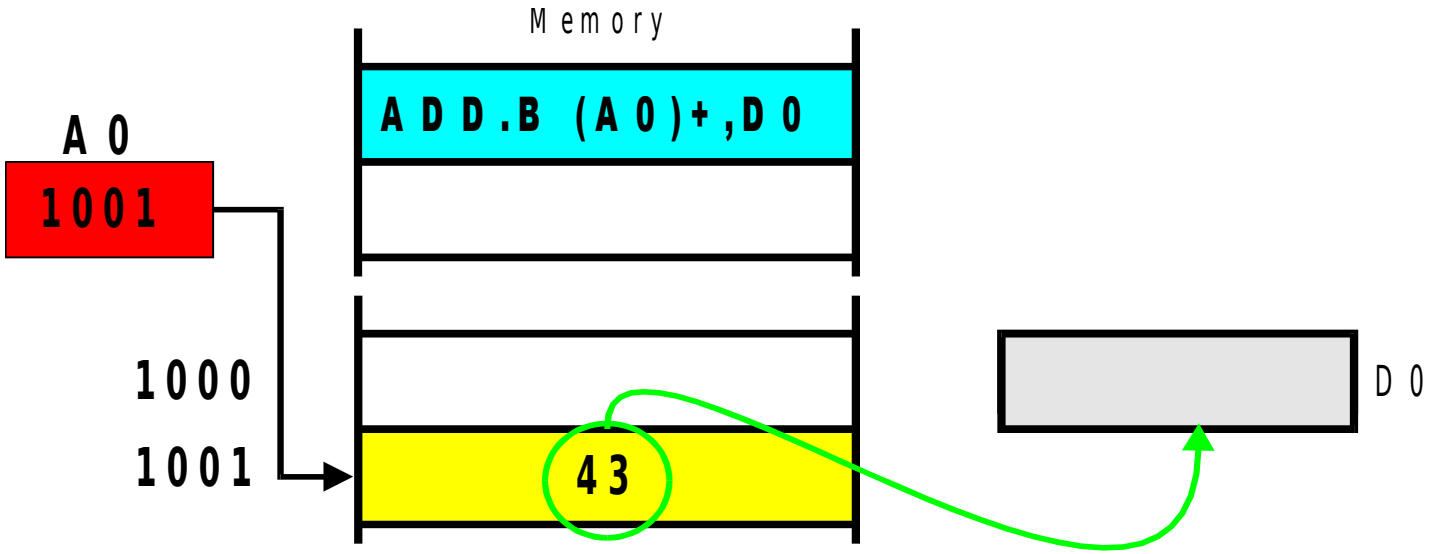
Il registro indirizzo contiene 1000
e punta alla locazione 1000

Auto-increment - Funzionamento



Il registro A0 viene usato per accedere alla locazione di memoria 1000 e il contenuto di questa locazione (57) viene sommato a D0

Auto-increment - Funzionamento



Dopo che l'istruzione è stata eseguita,
il contenuto di A0 viene incrementato, per
puntare alla locazione successiva

Problema

- Scrivere un programma assembly che sommi cinque numeri memorizzati in locazioni di memoria consecutive
- Assemblare ed eseguire sul simulatore
- Sperimentare:
  - L'effetto dell'istruzione LEA
  - L'effetto dell'autoincremento

Esempio

Il seguente frammento di codice usa l'address register indirect addressing con post-increment per sommare cinque numeri memorizzati in locazioni di memoria consecutive.

```
        MOVE.B  #5,D0       Cinque numeri da sommare
        LEA     Table,A0    A0 punta ai numeri
        CLR.B   D1          Inizializza la somma
Loop    ADD.B   (A0)+,D1    Somma il numero al totale
        SUB.B   #1,D0
        BNE     Loop        Fino a sommare tutti i numeri
        STOP    #$2700
        *
Table   DC.B    1,4,2,6,5   Dati d'esempio
```

Seguiamo il trace del programma, istruzione per istruzione

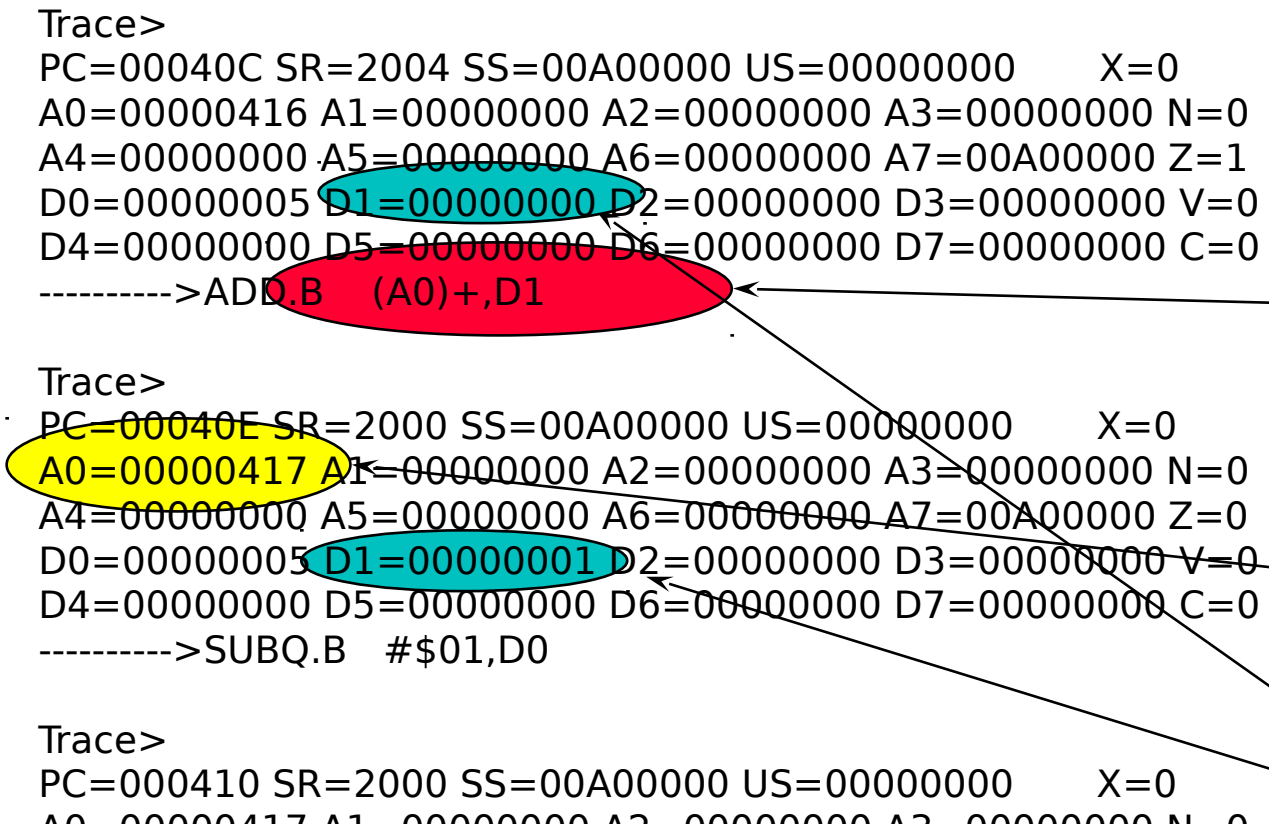
```
>DF
PC=000400 SR=2000 SS=00A00000 US=00000000      X=0
A0=00000000 A1=00000000 A2=00000000 A3=00000000 N=0
A4=00000000 A5=00000000 A6=00000000 A7=00A00000 Z=0
D0=00000000 D1=00000000 D2=00000000 D3=00000000 V=0
D4=00000000 D5=00000000 D6=00000000 D7=00000000 C=0
---------->MOVE.B   #$05,D0

>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
```

La prima istruzione carica D0 con il valore del literal 5

D0 è stato caricato con 5

Questa istruzione caria A0 con il vaòore $0416

A0 contiene $0416



Questa istruzione carica il contenuto della locazione puntata da A0 in D1

Siccome l'operando era (A0)+, il contenuto di A0 viene incrementato

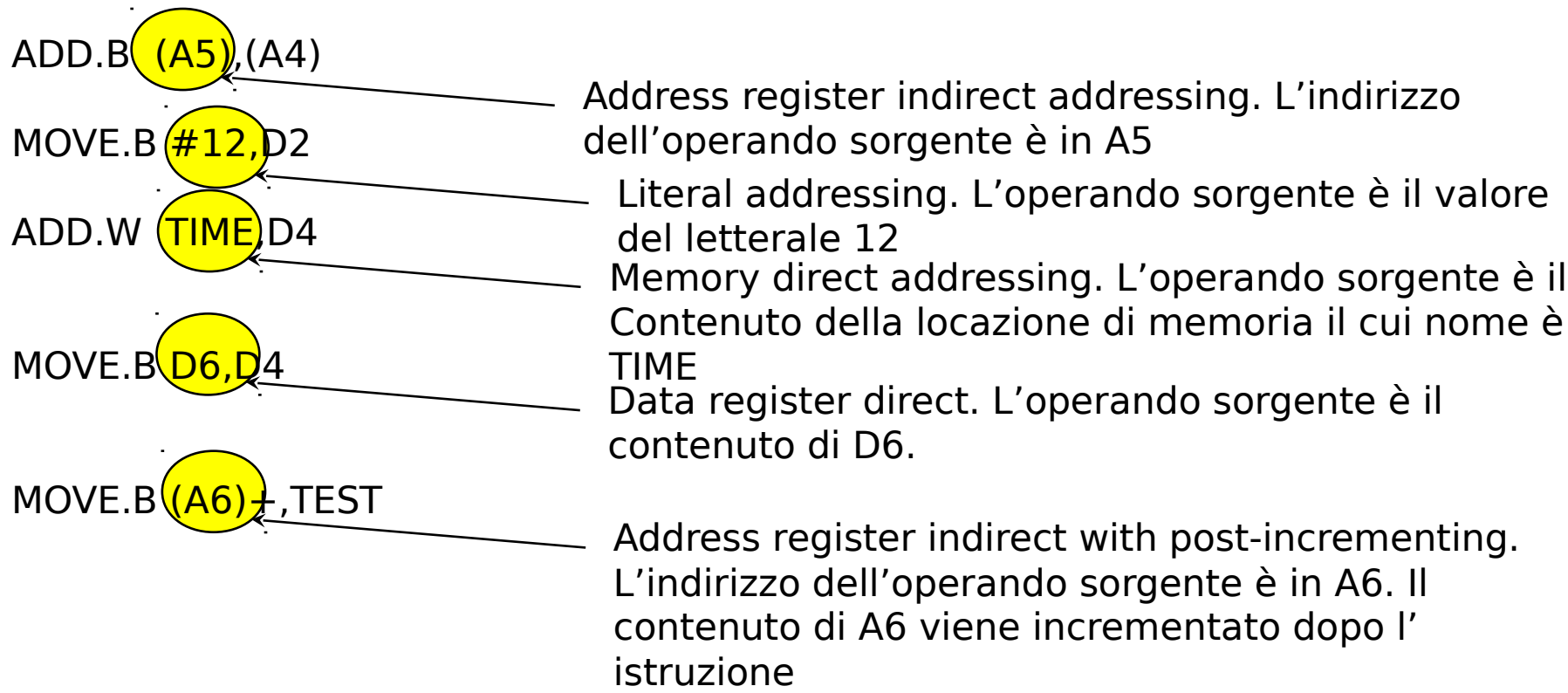
ADD.B (A0)+,D1
somma l'operando sorgente a D1

```
Trace>
PC=00040C 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
---------->ADD.B    (A0)+,D1

Trace>
PC=00040E SR=2000 SS=00A00000 US=00000000      X=0
A0=00000418 A1=00000000 A2=00000000 A3=00000000 N=0
A4=00000000 A5=00000000 A6=00000000 A7=00A00000 Z=0
D0=00000004 D1=00000005 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=00000418 A1=00000000 A2=00000000 A3=00000000 N=0
A4=00000000 A5=00000000 A6=00000000 A7=00A00000 Z=0
D0=00000003 D1=00000005 D2=00000000 D3=00000000 V=0
D4=00000000 D5=00000000 D6=00000000 D7=00000000 C=0
---------->BNE.S    $040C
```

Al ciclo successivo l' istruzione ADD.B (A0)+,D1 usa A0 come operando sorgente e poi incrementa il contenuto di A0

Problema

Identificare l'addressing mode usato per l'operando sorgente
In ciascuna delle seguenti istruzioni



Indirizzamento indiretto con scostamento

L'operando a cui accediamo sta all'indirizzo ottenuto dalla somma tra l'indirizzo nel registro indirizzi e lo scostamento a 16 bit.

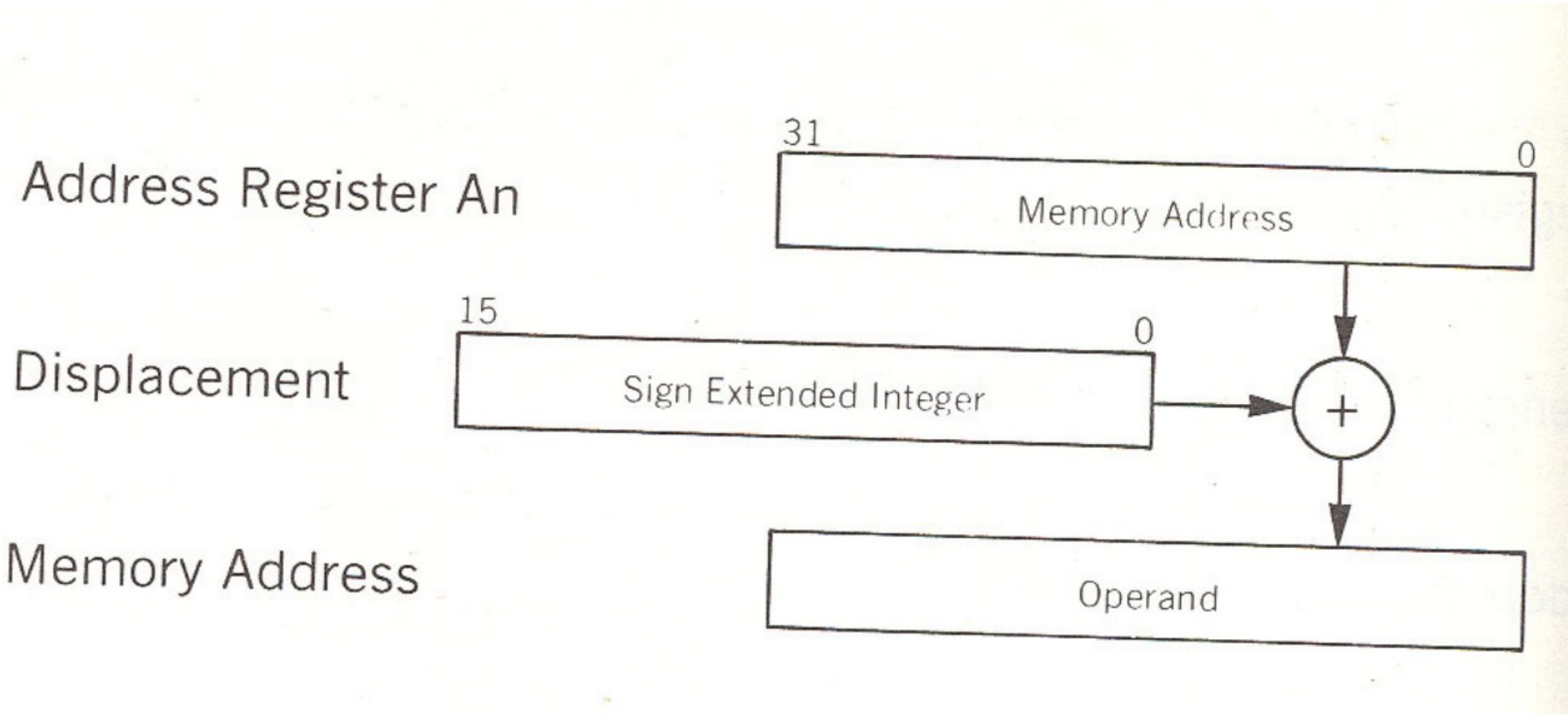
```
EQU         SCOSTAMENTO   10
            ADD.B         10 (A4),D5
            ADD.B         SCOSTAMENTO (A4),D5
```

L'istruzione tipicamente serve ad accedere ai campi di una struttura dati. Supponiamo ad es. che un codice C dichiari una variabile così:

```
struct { int Num;   char id1;  char id2; } pippo;
```

e che debba accedere ai vari campi così:

```
pippo.Num = 10;
pippo.id2 = 'c';
```



• **SCOSTAMENTO** è una costante direttamente inserita o è una label definita mediante la direttiva EQU. Viene utilizzato come espresso in complemento a due, quindi può essere anche un valore negativo.

```
NUM       EQU       0
ID1       EQU       2
ID2       EQU       3

          ORG       $1000
PIPPO:    DS.B      4

          ORG       $4000
START:    LEA       PIPPO,A0
          MOVE.W    #10,NUM(A0)
          MOVE.B    #'c',ID2(A0)

FINISH:   STOP      #$2700
          END       START
```

Indirizzamento con scostamento e indice

```
EQU     SCOSTAMENTO   10
        ADD.B     10 (A4,D1),D5
        ADD.B     SCOSTAMENTO (A4,D1),D5
```

La forma più generale di questo indirizzamento è quindi:

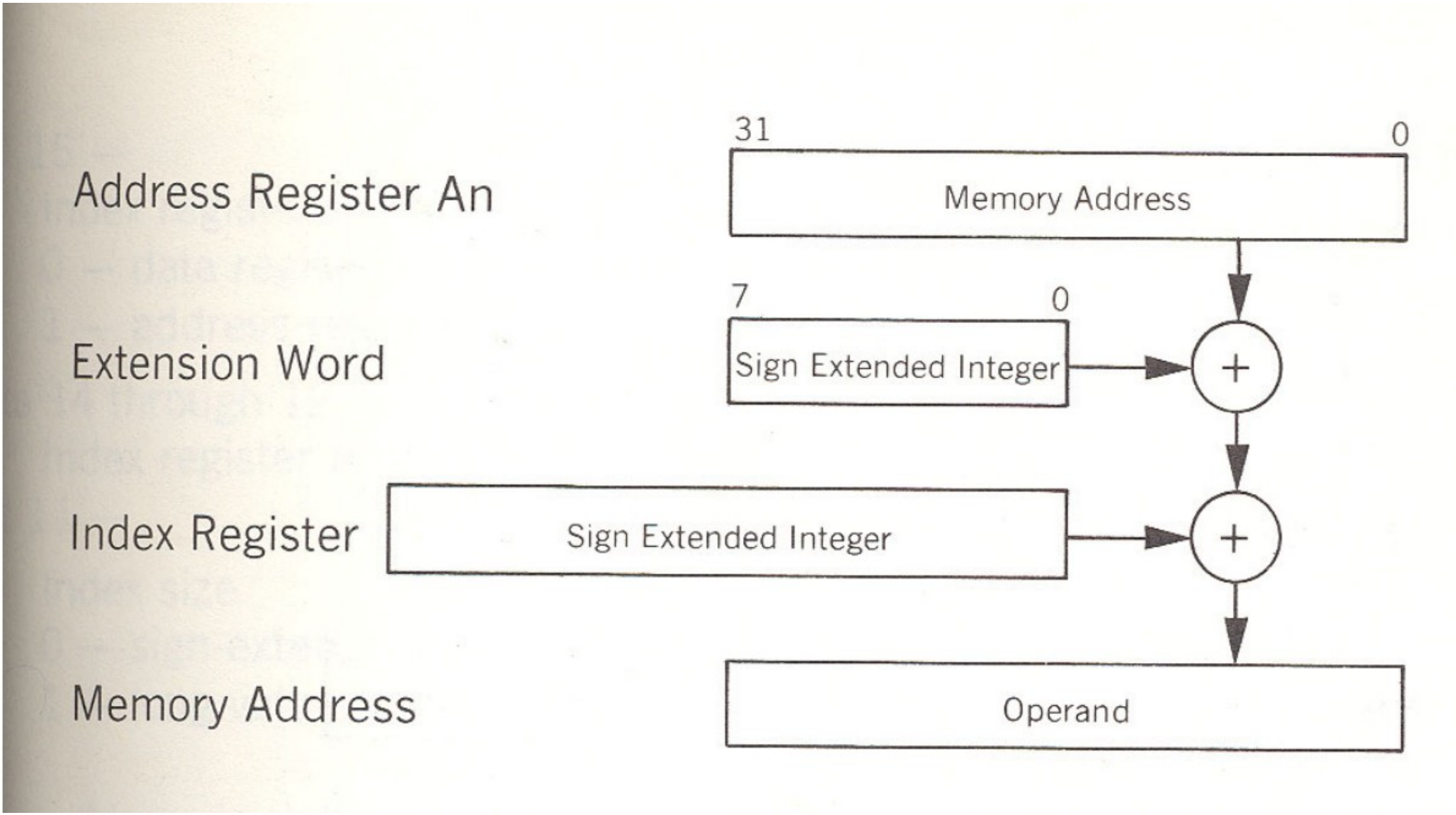
SCOSTAMENTO (An,Rn)

• **An** è un registro indirizzi,
• **Rn** è o un registro indirizzi o un registro dati, per default viene usato la word meno significativa (.W) se viene aggiunto il postfisso .L (D1.L) viene invece utilizzato tutto il registro a 32 bit.
• **SCOSTAMENTO** è una costante direttamente inserita o è una label definita mediante la direttiva EQU. Viene utilizzato come espresso in complemento a due, quindi può essere anche un valore negativo.

Allora l'indirizzo del dato varrà:

Indirizzo del dato = An + Rn + SCOSTAMENTO

Indirizzamento con scostamento e indice



Indexed

- In generale, l'Indexed Addressing combina due componenti mediante somma, per formare l'EA
  - Il primo componente è detto *base address* ed è specificato come parte dell'istruzione (come nell'absolute addressing)
  - Il secondo componente è detto *index register* e contiene il valore da sommare al base address per ottener l'EA
- È adatto per accedere ai valori di array e di tabelle
- Il processore MC68000 non supporta esplicitamente l'Indexed Addressing. Tuttavia, è possibile usare l'Indexed Short Addressing nei (32+32)Kbyte agli estremi dei 4GB dello spazio di memoria
- Esempio:

```
MOVEA.W     I,AO
MOVE.B      CLIST-1(AO),D1          Leggi clist[i]
```

Accesso agli elementi di un vettore

```
            ORG     $8100
NUMEL       EQU     3

VETT        DS.L    NUMEL
INDEX       DC.W     2

START       LEA         VETT,A3
            MOVE.W       INDEX,D1
            MULU        #4,D1
            MOVE.L      0(A3,D1),D4
            STOP        #$00
            END         START
```

Based Addressing

- Based Addressing è esattamente l'inverso dell'Indexed Addressing, in quanto combina due componenti mediante somma, per formare l'EA, ma:
  - Il primo componente è detto *displacement* ed è specificato come parte dell'istruzione (come nell'absolute addressing)
  - Il secondo componente è detto *base address* ed è contenuto in un registro
- È adatto per accedere ai valori di array e di tabelle di cui si conosca la posizione relativa ad assembly time, ma non quella iniziale
- Il processore MC68000 supporta il Based Addressing come l'Indexed

Based Indexed

- Based Indexed Addressing combina due componenti mediante somma, per formare l'EA, ma:
  - Il primo componente è detto registro base e contiene il *base address*
  - Il secondo componente è detto registro indice e contiene il *displacement*
- Consente di calcolare a run time sia la posizione iniziale che quella relativa di tabelle ed array
- Il processore MC68000 supporta lo Short Based Indexed ed il Long Based Indexed

Relative Addressing

- (Relative to the PC)
- Questi modi di indirizzamento calcolano l'indirizzo effettivo come la somma di un displacement fisso specificato nell'istruzione e del valore corrente del PC
- Fanno spesso uso di displacement piccoli, di 8 o 16 bit, per specificare indirizzi vicini all'istruzione corrente, anziché ricorrere a indirizzi assoluti di 32 bit
- Il 68000 non consente di utilizzare questi modi di indirizzamento per specificare operandi che potrebbero essere modificati

Relative Indexed Addressing

- Variante del Relative
- Funziona come il Based Indexed, ma il base register è sostituito dal PC
- Può essere usato per saltare ad aree di memoria read-only, contenenti dati o istruzioni