Corso di Calcolatori Elettronici I
A.A. 2012-2013

Modi di indirizzamento

ing. Alessandro Cilardo

Accademia Aeronautica di Pozzuoli
Corso Pegaso V “GArn Elettronici”

Modi di indirizzamento

- Indicano come la CPU accede agli operandi usati dalle proprie istruzioni
- La loro funzione è quella di fornire un indirizzo effettivo (*Effective Address*, 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.
  - il 68000 ne supporta una buona parte

Modi di Indirizzamento del 68K

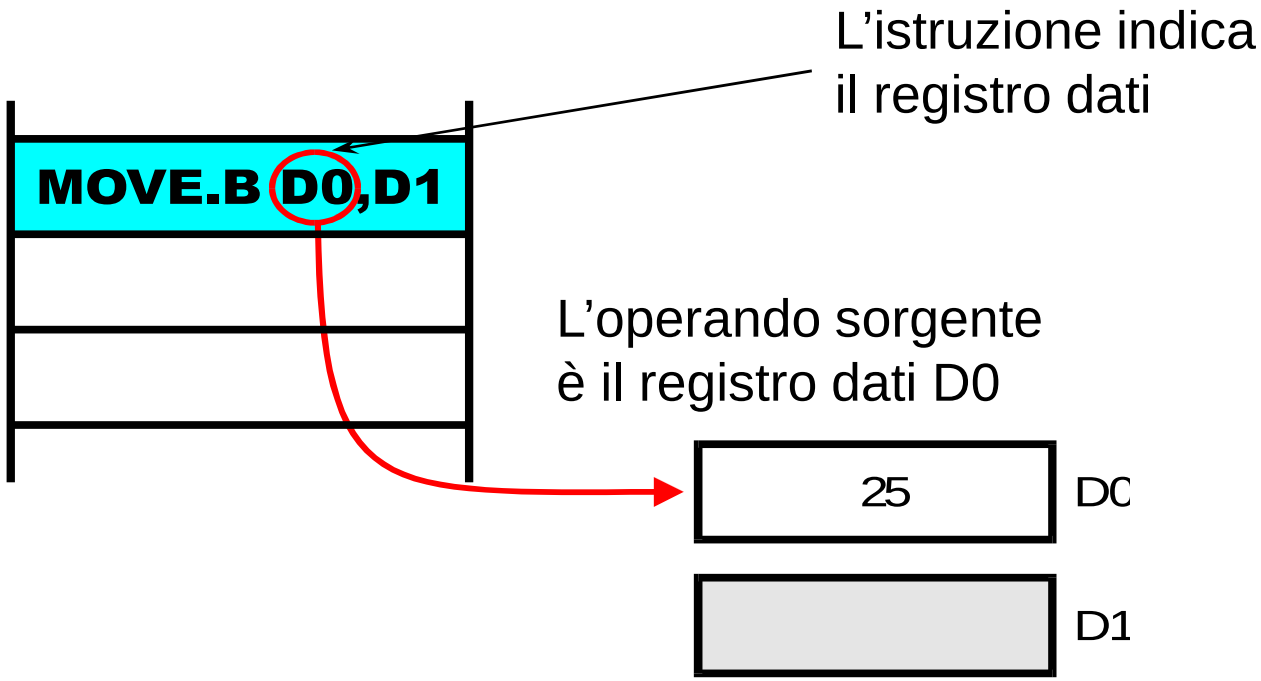
- Register Direct
  - Data-register Direct
  - Address-register Direct
- Immediate (or Literal)
- Absolute
  - short
  - long
- Address-register Indirect
- Auto-Increment
- Auto-Decrement
- Indexed short
- Based
- Based Indexed
  - short
  - long
- Relative
- Relative Indexed
  - short
  - long

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, sovrascrivendo il contenuto precedente

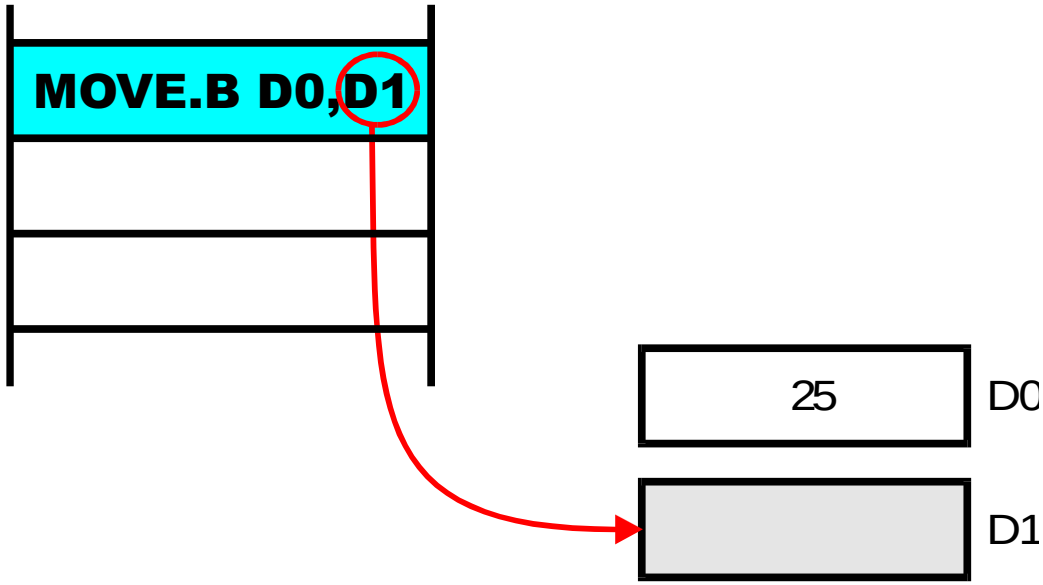
```
MOVE.B  D0,D3   Copia l'operando sorgente in D0 nel  registro D3
SUB.L   A0,D3   Sottrae l'operando sorgente nel registro A0 dal registro D3
CMP.W   D2,D0   Confronta l'op. sorgente nel registro D2 con il registro D0
ADD     D3,D4   Somma l'operando sorgente nel registro D3 al registro D4
```

Register Direct Addressing: Funzionamento



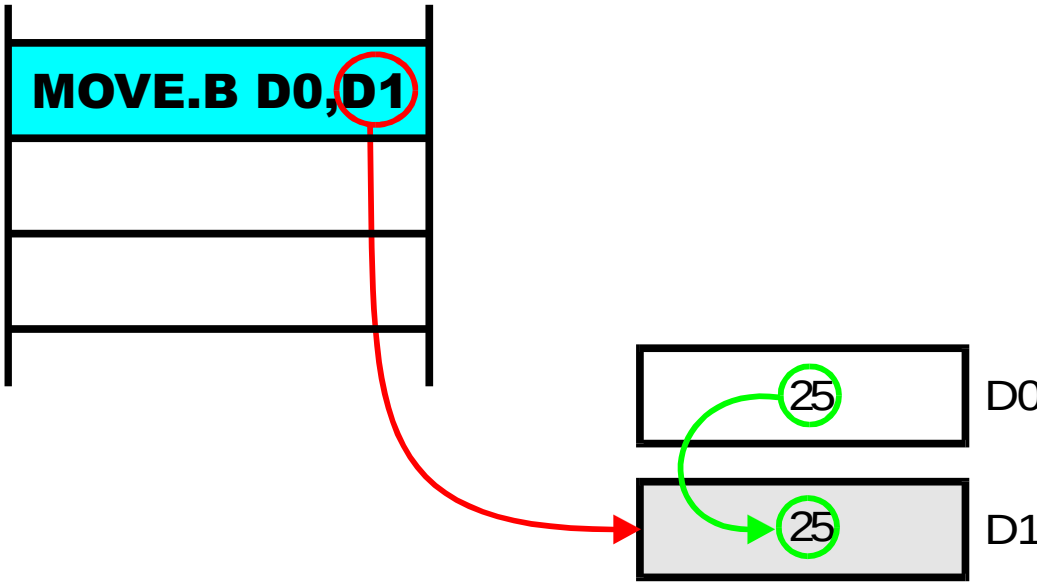
L'istruzione **`MOVE.B D0,D1`** usa registri
dati sia per l'operando sorgente che per quello destinazione

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 dati 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 (*reg*), più un bit per indicare quale gruppo di registri usare (*mode*)
  - *mode* = 0, *reg* = 0-7 per Dn
  - *mode* = 1, *reg* = 0-7 per An
  - Ad esempio, per codificare la MOVE D0,D1 bastano 16 bit di parola codice (non sono necessarie parole aggiuntive)
- I programmatori lo usano per memorizzare variabili che sono usate di frequente (*scratchpad storage*)

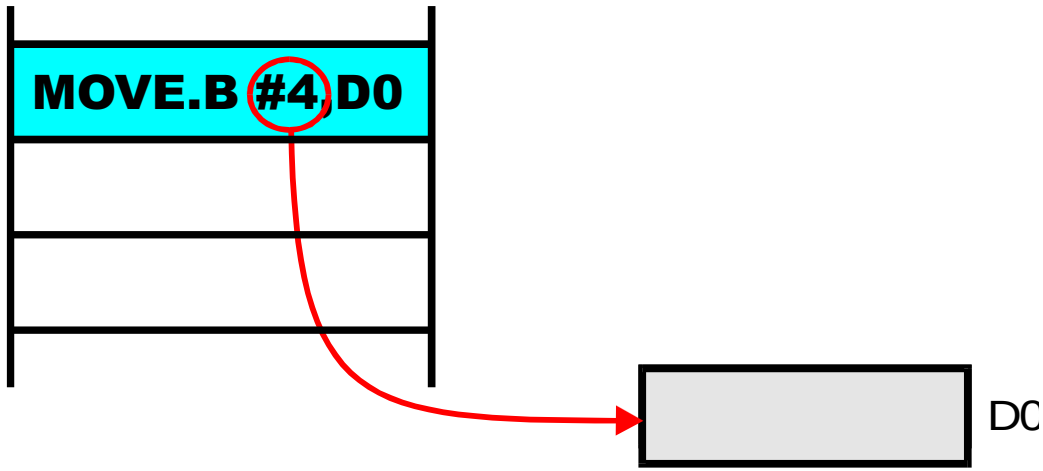
Immediate Addressing

- Il valore stesso dell'operando effettivo costituisce **parte dell'istruzione**
- Può essere usato unicamente per specificare un operando sorgente

  (non si può scrivere su una costante!)
- È indicato da un simbolo **#** davanti all'operando sorgente
- Un operando immediato è anche chiamato *literal*

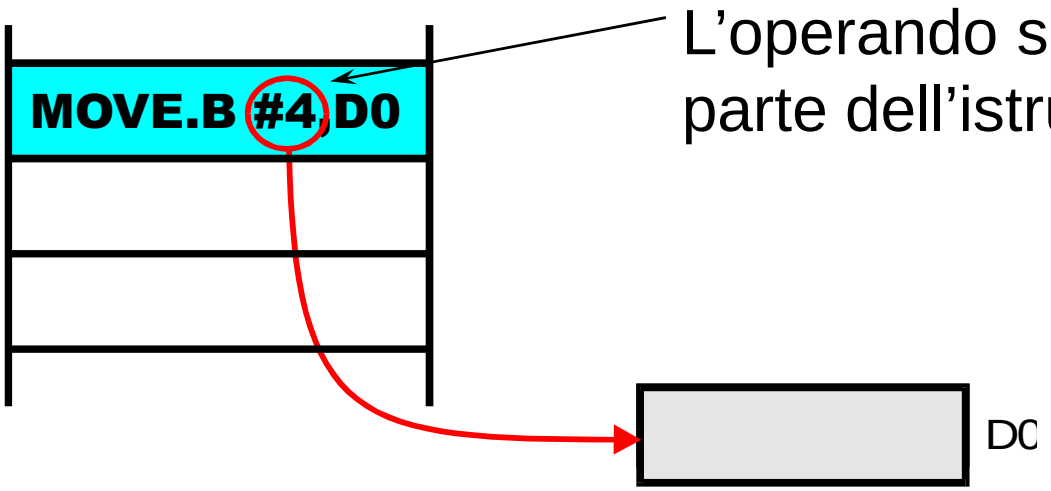
```
MOVE.B #4,D0  Usa l'operando sorgente immediato 4
```

Immediate Addressing: Funzionamento

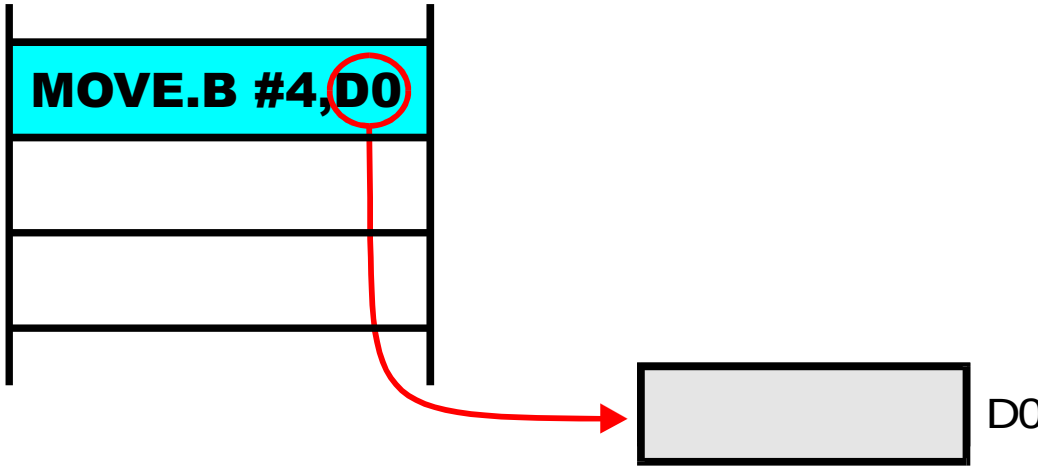


L'istruzione **`MOVE.B #4,D0`** usa un operando sorgente *literal* ed un operando destinazione *register direct*

Immediate Addressing: Funzionamento

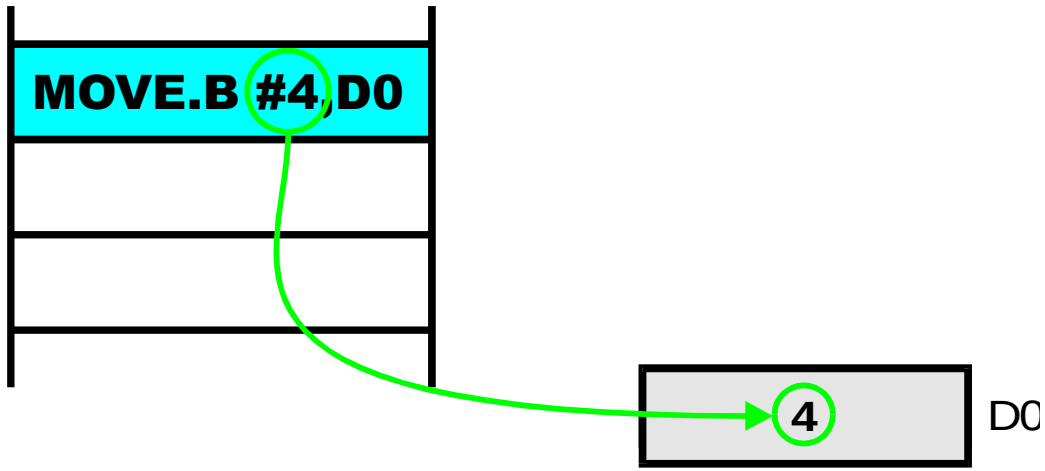


Immediate Addressing: Funzionamento



L’operando destinazione è un registro dati

Immediate Addressing: Funzionamento



L'effetto di questa istruzione è quello di copiare il *valore* 4 nel registro dati D0

Immediate Addressing: Caratteristiche

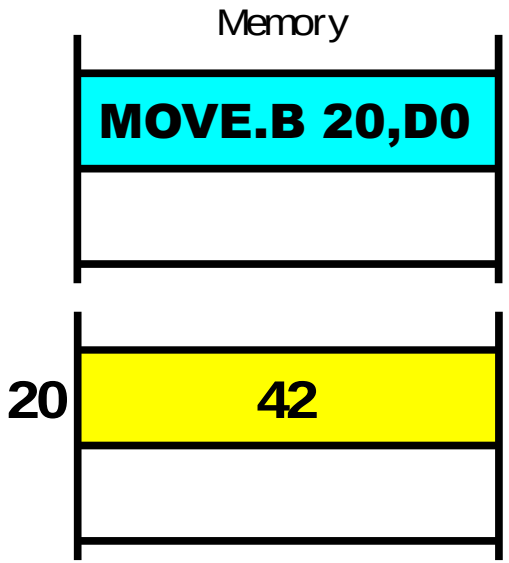
- Se la costante da manipolare ha dimensioni ridotte (pochi bit) è possibile codificarla direttamente nei 16 bit dell'istruzione
  - non sono necessarie parole aggiuntive per codificare il *literal* oltre alla parola codice di 16 bit
  - se l'operando destinazione è un registro (*register direct addressing*), l'intera istruzione viene codificata su 16 bit
  - non sono necessarie ulteriori (lenti) accessi in memoria
- Se la costante è "lunga", è necessario usare una o più parole aggiuntive che seguono la parola codice

Absolute Addressing (o Direct Addressing)

- È il modo più semplice per specificare un indirizzo di memoria completo
- L'istruzione **fornisce direttamente il valore dell'indirizzo** dell'operando in memoria
- Richiede due accessi in memoria:
  - Il primo è per prelevare l'istruzione e l'indirizzo assoluto che essa fornisce
  - Il secondo è per accedere all'operando effettivo

```
CLR.B $1A3B  azzera il contenuto della locazione di memoria 1A3B
```

Absolute Addressing: Funzionamento

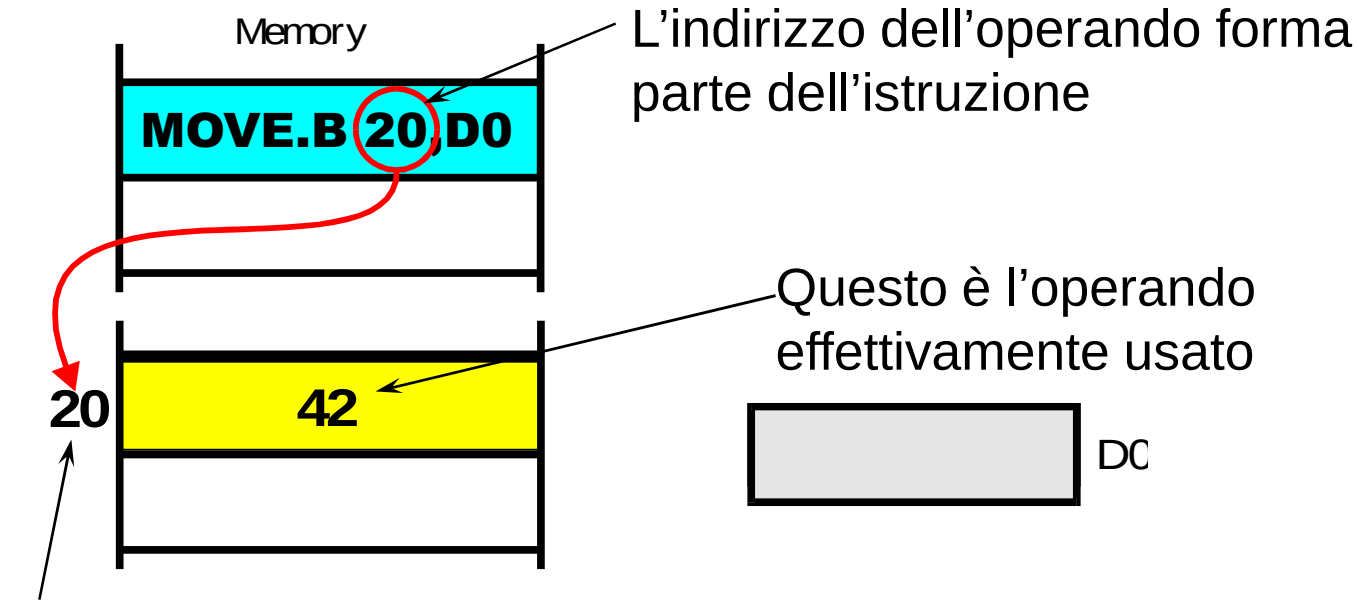


Questa istruzione ha un operando *absolute*

L'operando sorgente è in memoria

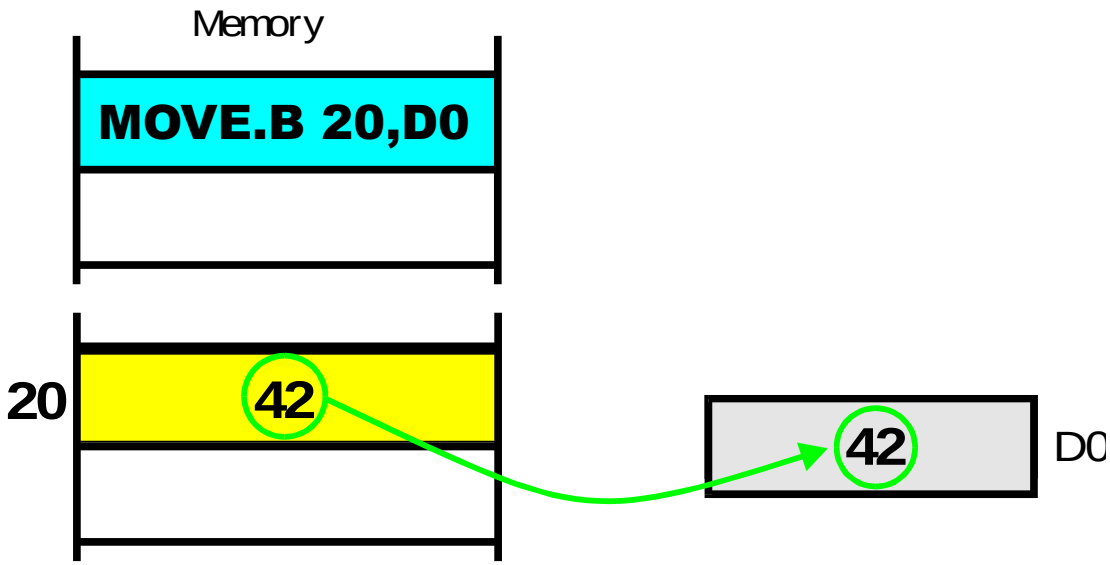
L'operando destinazione usa il *direct addressing* per un registro dati

Absolute Addressing: Funzionamento



Una volta che ha letto l'indirizzo dell'operando dall'istruzione, la CPU accede all'operando

Absolute Addressing: Funzionamento



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

Esempio modi fondamentali

- Consideriamo questo frammento di programma in linguaggio di alto livello (ad esempio C/C++):

```
char z, y = 27;
z = y + 24;
```

Il seguente frammento di codice lo implementa in assembler:

```
        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
```

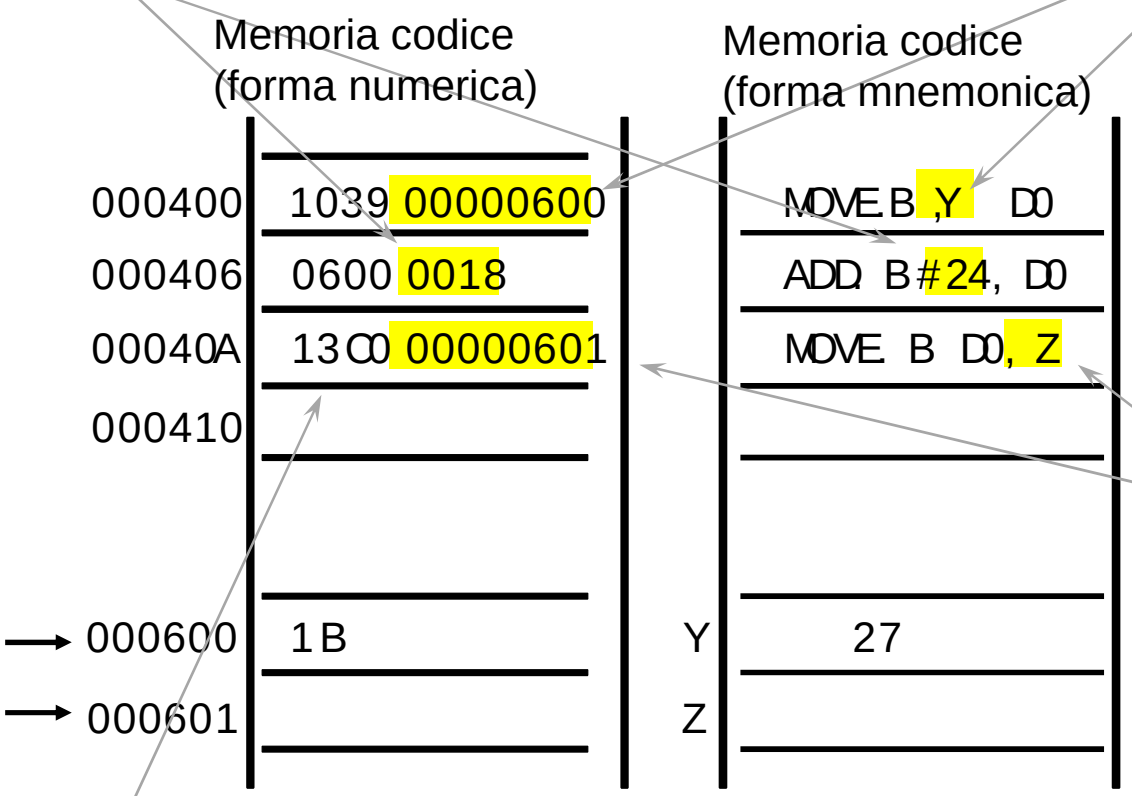
Esempio: il Programma Assemblato

```
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    00000600                          ORG     $600
6    00000600 1B               Y       DC.B    27
7    00000601 00000001         Z       DS.B    1
```

Esempio: Mappa della Memoria

#24 è un operando di tipo literal, memorizzato come parte dell'istruzione

Y = 600 è una variabile acceduta tramite direct addressing (000600)



Z = 601 è una a variabile acceduta mediante direct addressing (000601)

16 bit che codificano l'istruzione (ad es. la MOVE)

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

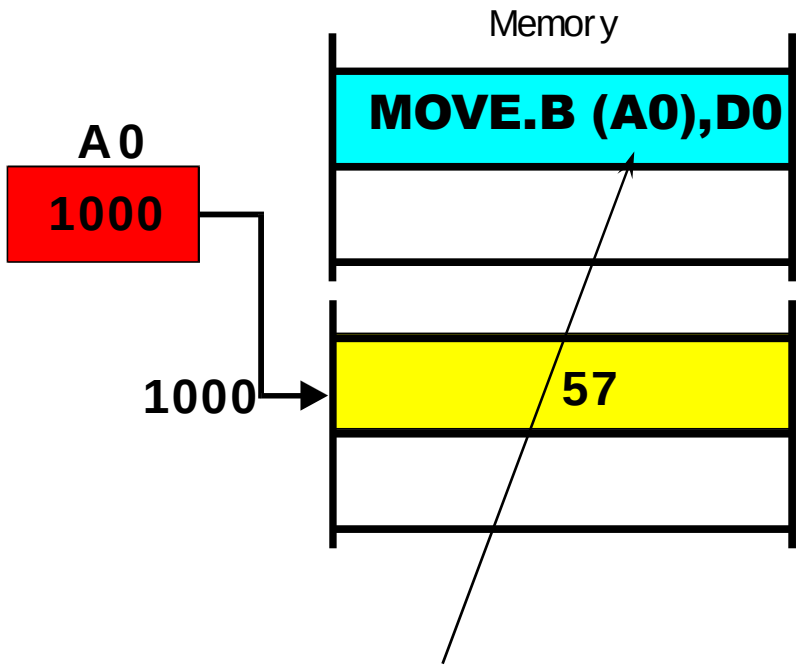
Address Register Indirect Addressing

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

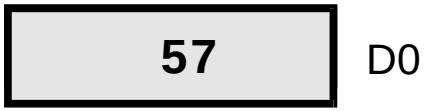
```
MOVE.B (A0),D0
```

```
Vai in memoria ad una posizione il cui
indirizzo è contenuto nel registro A0,
preleva un byte e copialo nel registro D0
```

Address Register Indirect: Funzionamento

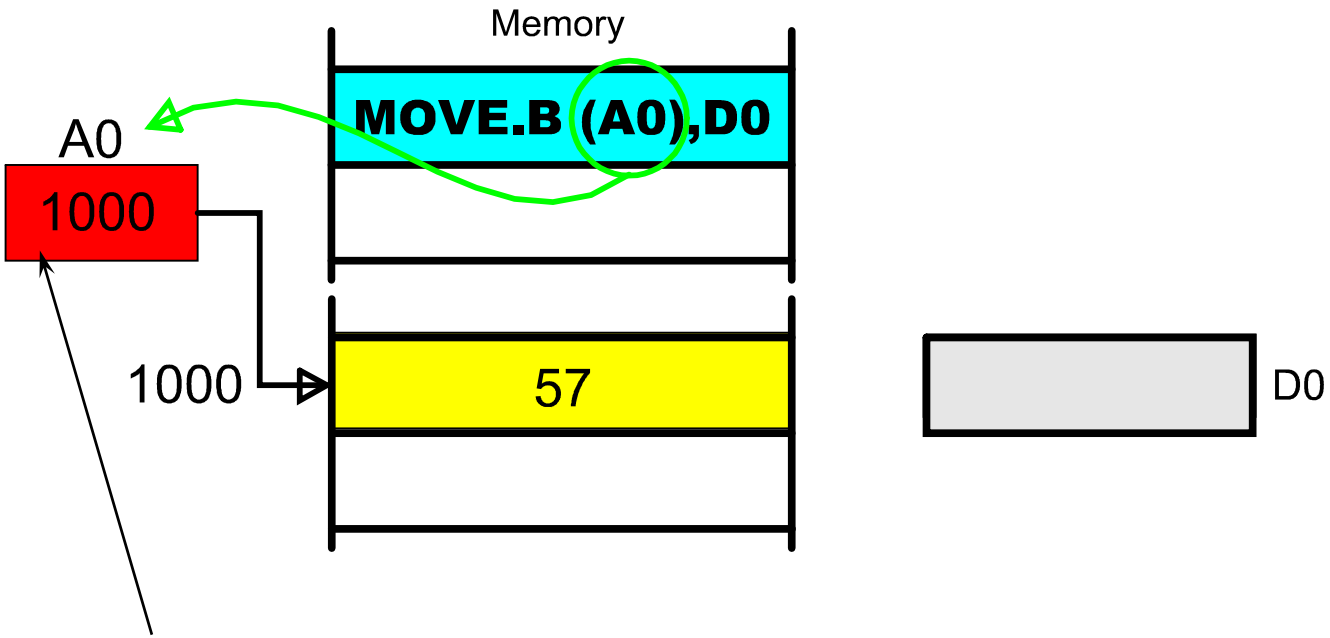


Questa istruzione significa: carica D0 con il contenuto della locazione puntata dal registro indirizzo A0



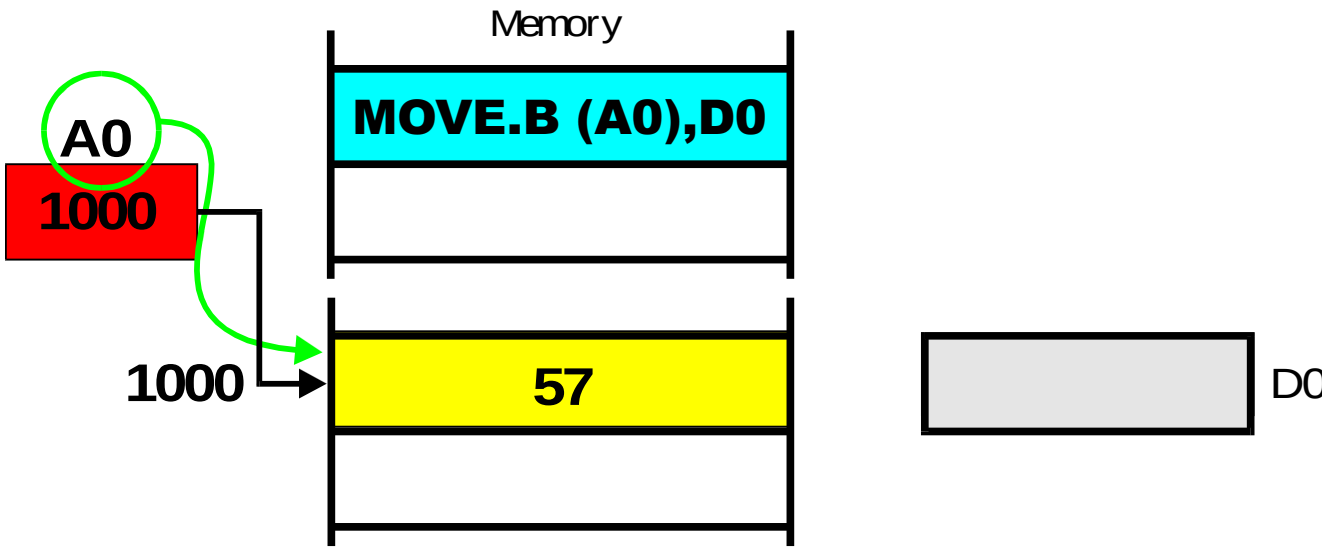
L'istruzione specifica l'operando sorgente come (A0)

Address Register Indirect: Funzionamento



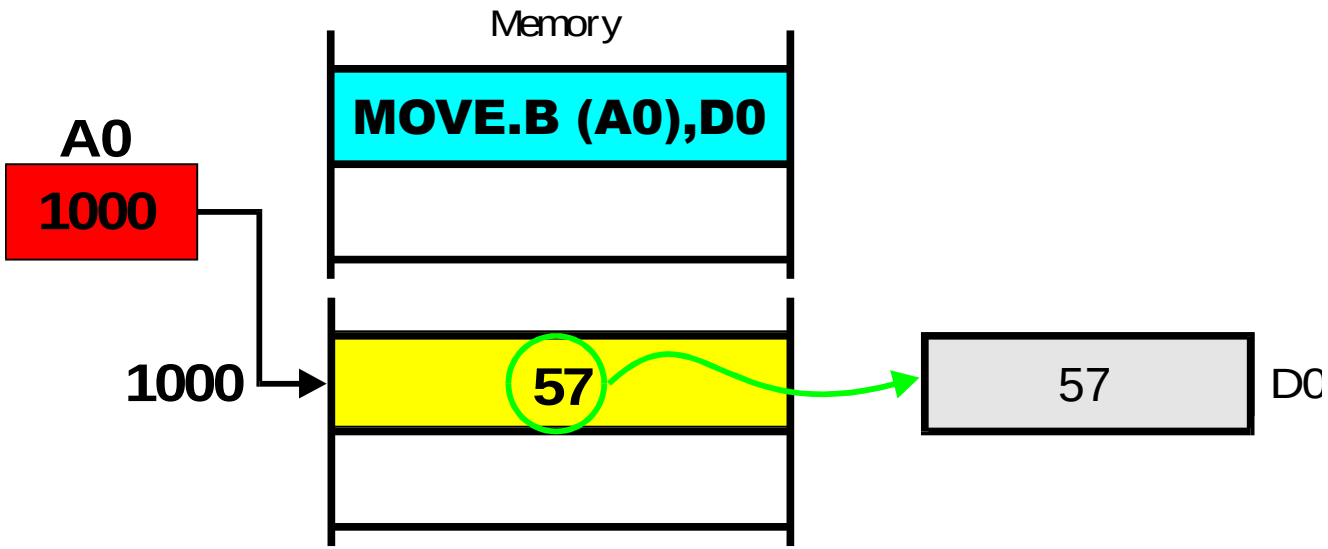
Il registro indirizzo nell'istruzione specifica un registro indirizzo che contiene l'indirizzo dell'operando

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, usando la modalità *Address Register Indirect*.
- Se il modo di indirizzamento è specificato come **(An)+**, il contenuto del registro indirizzo è **incrementato automaticamente** di una quantità pari alla dimensione dell'operando *dopo* *l'uso* ("post-incremento")

```
MOVE.W (A0)+, D0
```

```
Usa A0 per la MOVE e poi gli aggiunge 2
(2 poiché l'accesso è di tipo .W = 2
byte). Di fatto, l'istruzione esegue un
pop in D0 dallo stack puntato da A0
```

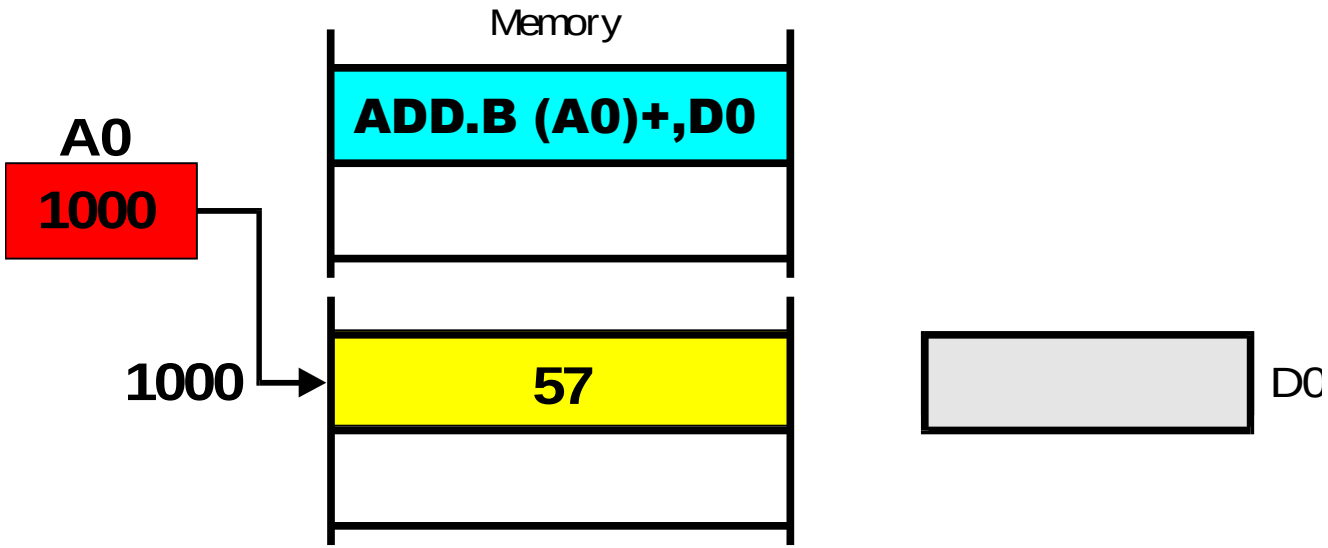
Auto-decrement

- L'istruzione specifica uno dei registri indirizzo, usando la modalità *Address Register Indirect*.
- Se il modo di indirizzamento è specificato come **-(An)**, il contenuto del registro indirizzo è **decrementato automaticamente** di una quantità pari alla dimensione dell'operando *prima dell'uso* ("pre-decremento")

```
MOVE.W D0,-(A0)
```

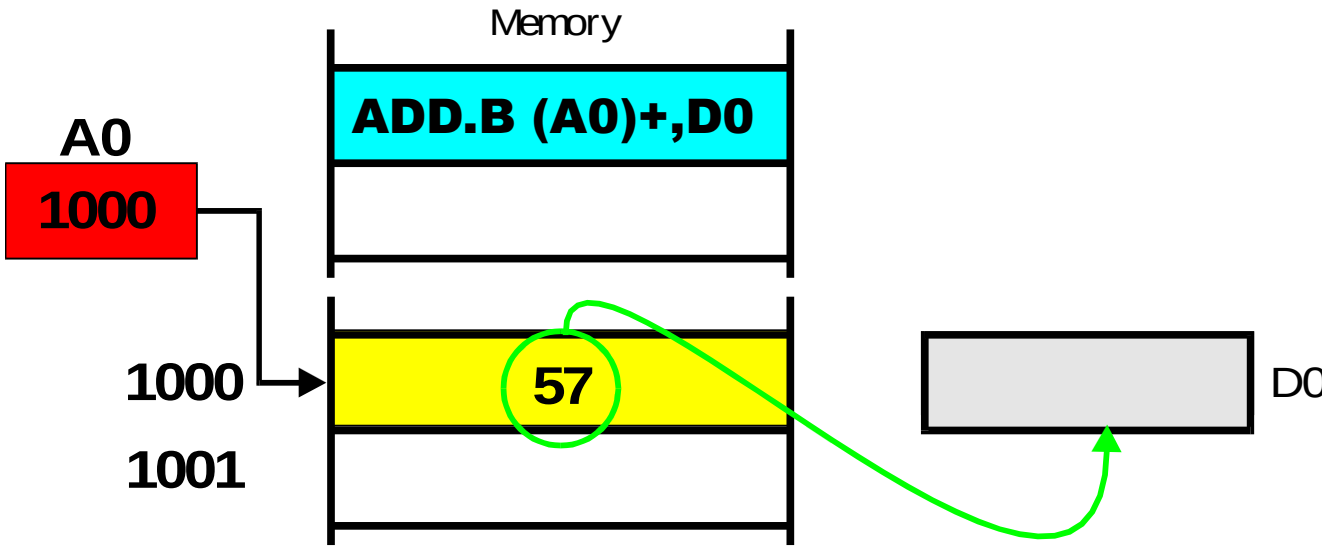
Sottrae 2 ad A0 e poi lo usa per la MOVE (2 poiché l'accesso è di tipo .W = 2 byte). Di datto, l'istruzione esegue un *push* di D0 sullo *stack* puntato da A0.

Auto-increment: Funzionamento



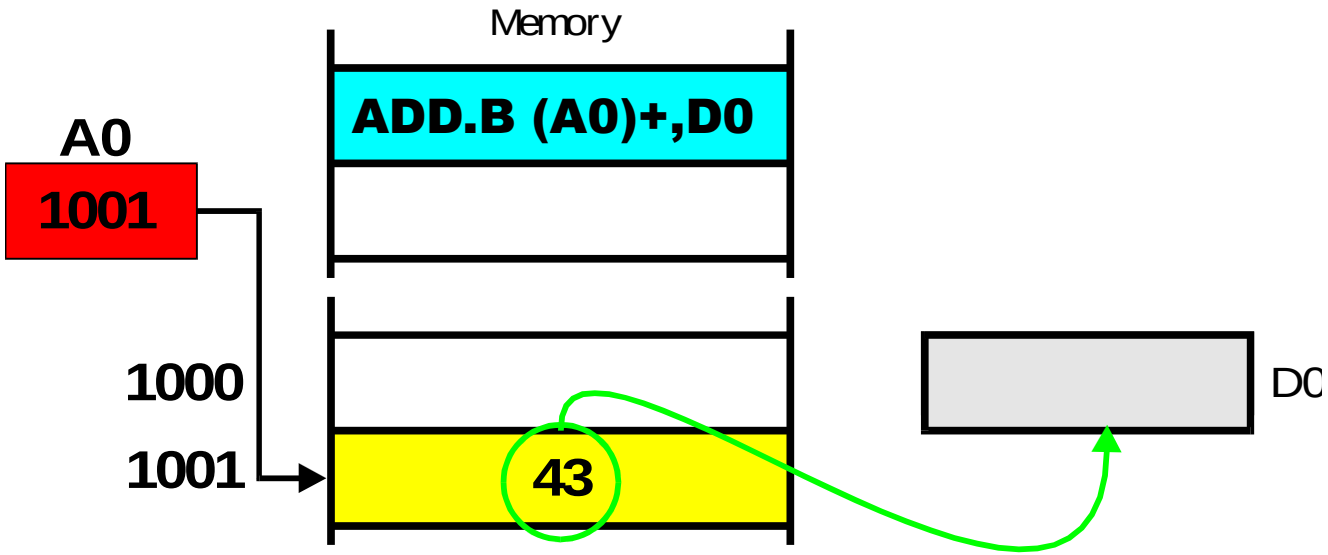
Il registro indirizzo contiene 1000
ovvero “punta” alla locazione 1000

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

Istruzione LEA

- Serve a calcolare e caricare in un registro **A*x*** l'*Effective Address* (EA) così come verrebbe calcolato da un'istruzione che abbia indirizzi di memoria come operandi
- E' utile per evitare di far calcolare direttamente all'istruzione l'EA, generandolo prima ed eventualmente salvandolo in un registro **A*x***
- Nel caso di indirizzi costanti (che non vanno quindi calcolati) non è particolarmente utile ed è sostituibile con un trasferimento dell'indirizzo (costante) nel registro destinazione **A*x***
  - (vedi esempio sotto)

```
LEA $0010FFFF,A5
```

è equivalente a

```
MOVE #$0010FFFF,A5
```

```
Sposta la costante esadecimale 0010FFFF nel
registro A5. NOTA: se avessimo usato una
MOVE in luogo della LEA (senza il #),
avremmo copiato il contenuto della locazione
di indirizzo 0010FFF nel registro A5
```

Esempio

- 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: Codice

- 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 alla tabella dei numeri
        CLR.B   D1          Inizializza la somma
Loop    ADD.B   (A0)+,D1    Somma il numero al totale
        SUB.B   #1,D0       Decrementa D0 per gestire il conteggio
        BNE     Loop        Itare fino a 5
Table   DC.B    1,4,2,6,5   Dati d'esempio
```

Esempio: tracce del programma

Traccia: 1/3

```
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

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

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
```

D0 è stato caricato con **5**

Questa istruzione carica **A0** con il valore **$0416**

A0 ora contiene **$0416**

Esempio: tracce del programma

Traccia: 2/3

```
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

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

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
```

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**

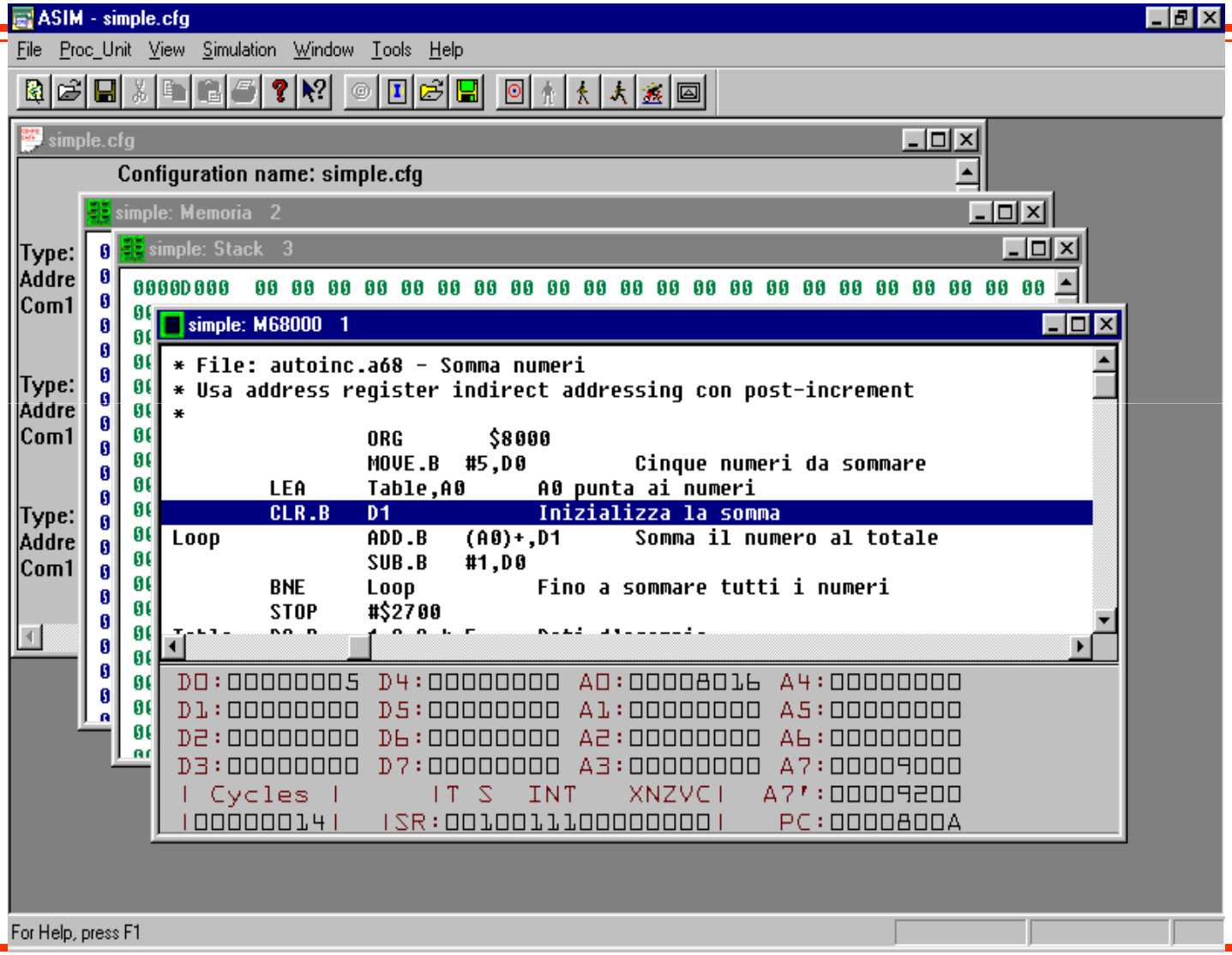
Esempio: tracce del programma

Traccia: 3/3

```
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

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

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**

Esempio: Esecuzione



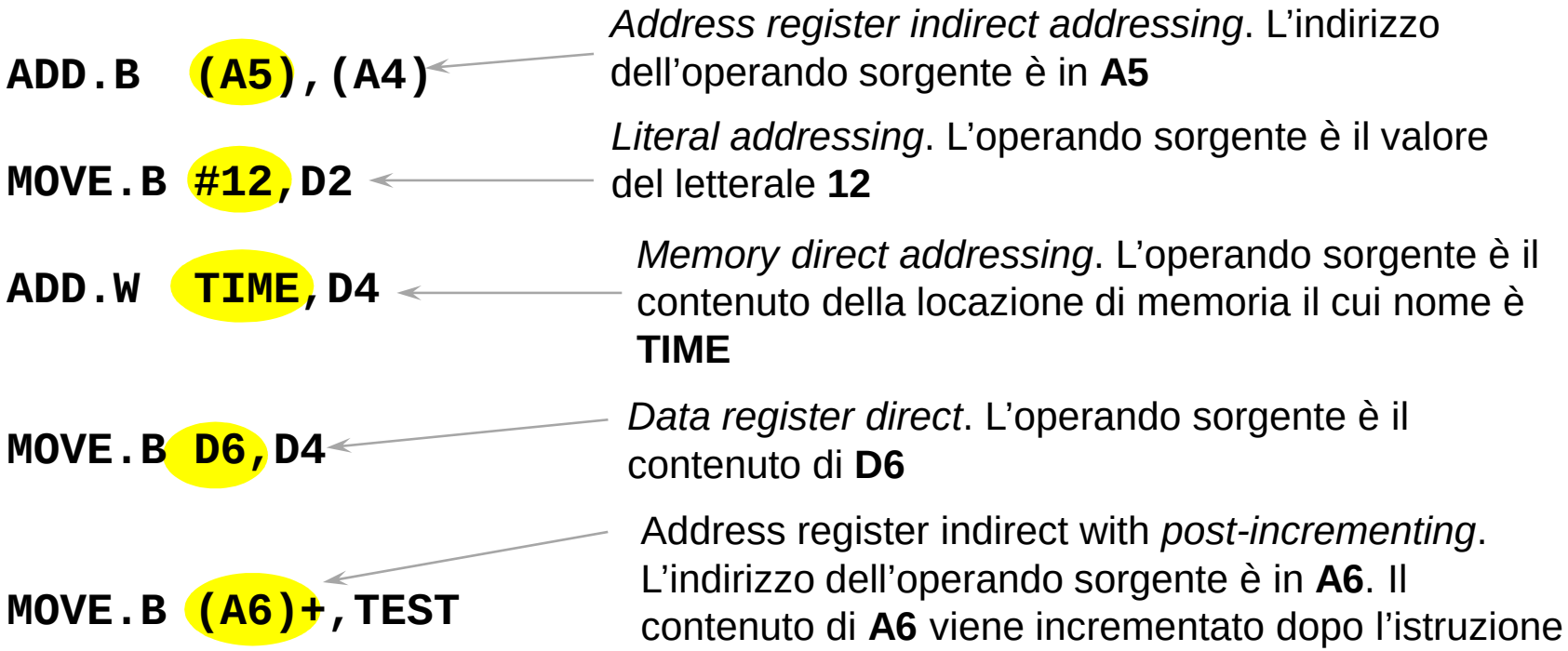
Problema

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

```
ADD.B  (A5),(A4)

MOVE.B #12,D2

ADD.W  TIME,D4


MOVE.B D6,D4


MOVE.B (A6)+,TEST
```

Problema: soluzione

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



Problema

- Se doveste tradurre in linguaggio assembly il seguente frammento di pseudo-codice, quali modi di indirizzamento utilizzereste?

```
SUM = 0

FOR J = 5 TO 19

  SUM = SUM + X(J)*Y(J)

END FOR
```

Problema

- Se doveste tradurre in linguaggio assembly il seguente frammento di pseudo-codice, quali modi di indirizzamento utilizzereste?

```
SUM = 0

FOR J = 5 TO 19

SUM = SUM + X(J)*Y(J)

END FOR
```

SUM è una variabile temporanea. La si può mettere in un registro ed usare il *register direct addressing*

J è una variable temporanea, che tipicamente viene messa in un registro e viene inizializzata al valore del *literal* **5**

X(J) è un elemento di un array, a cui tipicamente si accede mediante *address register indirect addressing*, eventualmente con *auto-incremento*

Indexed Addressing

- In generale, l'*Indexed Addressing* combina due componenti mediante somma, per formare l'Effective Address (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

```
MOVE.B  VET(A0),D1
```

VET è una costante definita nel programma assembly. La MOVE accede all'indirizzo VET sommato al contenuto del registro A0. Incrementando A0 si può accedere a locazioni successive in memoria

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

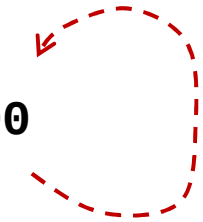
- *Based Indexed Addressing* combina due componenti mediante somma, per formare l'EA, ma:
  - Il primo componente è detto registro base e contiene il ***base address***
  - Il secondo componente è detto registro indice e contiene il ***displacement***
- Consente di calcolare a run time sia la posizione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

```
ADD.W $1C(A3,D2),D0   Accede alla memoria all'indirizzo calcolato
                      come somma del contenuto del registro A3,
                      del contenuto del registro D2 e della
                      costante esadecimale 1C. Somma il dato letto
                      dalle memoria e scrive il risultato in D0.
```

Esempio di uso della LEA

- Osservazione: Se l'istruzione **ADD** dell'esempio precedente dovesse essere ripetuta molte volte, sarebbe inefficiente ripetere ogni volta durante l'esecuzione il calcolo dell'Effective Address
- Conviene usare l'istruzione **LEA** per calcolare una sola volta l'Effective address e salvarlo in un registro **A*x***, che può poi essere utilizzato continuamente senza ripere il calcolo

```
LEA $1C(A3,D2),A5
   .
   .
   .
ADD.W (A5),D0
```



Calcola la somma del contenuto del registro A3, del contenuto del registro D2 e della costante esadecimale 1C. Scrive il risultato nel registro indirizzo A5.

Usa (eventualmente in maniera ripetuta, l'indirizzo in A5 senza ricalcolarlo

Relative Addressing

- "*Relative*" indica che il calcolo dell'indirizzo è relativo al Program Counter (PC)
  - ovvero, calcolato per **differenza rispetto all'indirizzo dell'istruzione attualmente eseguita**
- Questi modi di indirizzamento calcolano l'indirizzo effettivo come 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*
  - può essere usato anche per i dati ma è più frequente trovarlo isato per le istruzioni