

F3: Assembler för 68000

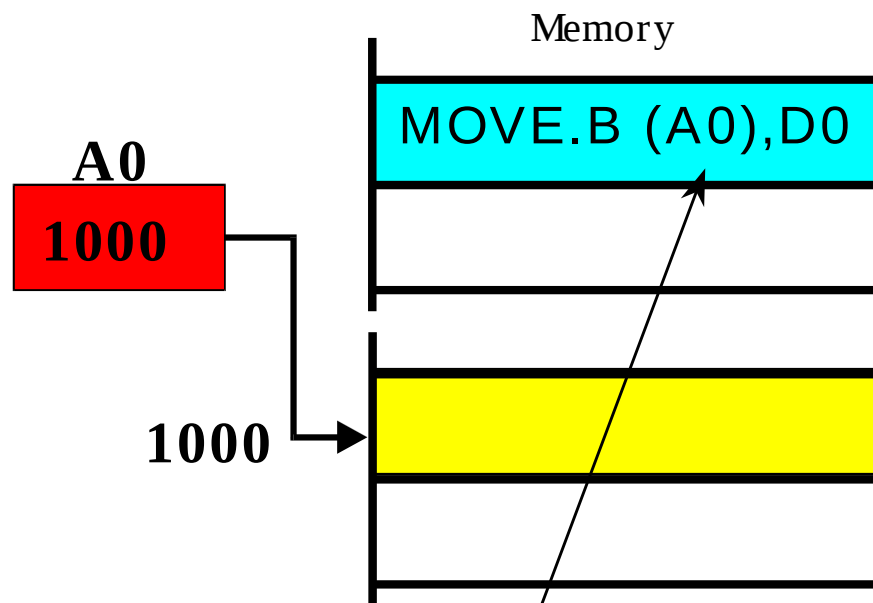
- **Adresseringsmetoder, fortsättning**
 - Indirektadressering
- **Pre-processor-kommandon**
- **Programmeringsmetodik**

Adresseringsmetoder

- Olika metoder
 - *Direkt register adressering*
 - *Direkt data adressering*
 - *Absolut adressering*
 - Indirekt adressering

Indirekt adressering via adressregister

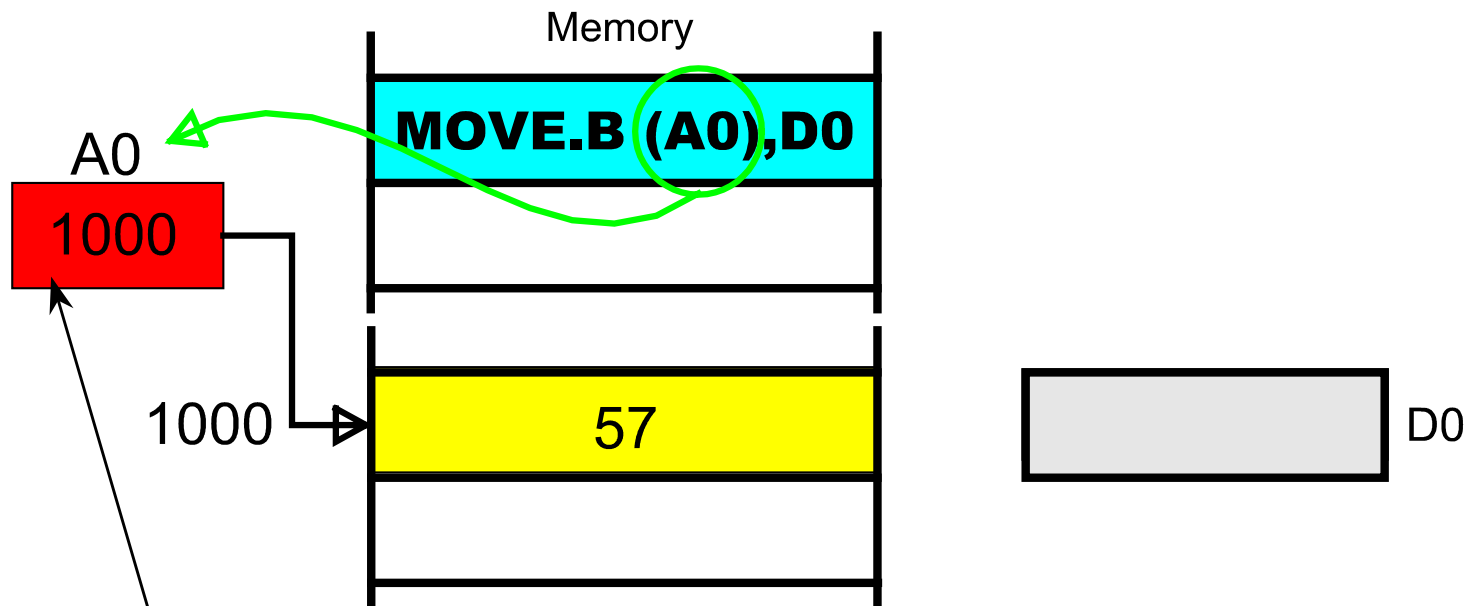
- Indirekt adressering, i instruktionen används ett av 68000s adressregister,
 - MOVE.B (**A0**),D0.
- Det specificerade adressregistret innehåller adressen till operanden.
- Processorn accessar sedan operanden som adressregistret **pekar på**.



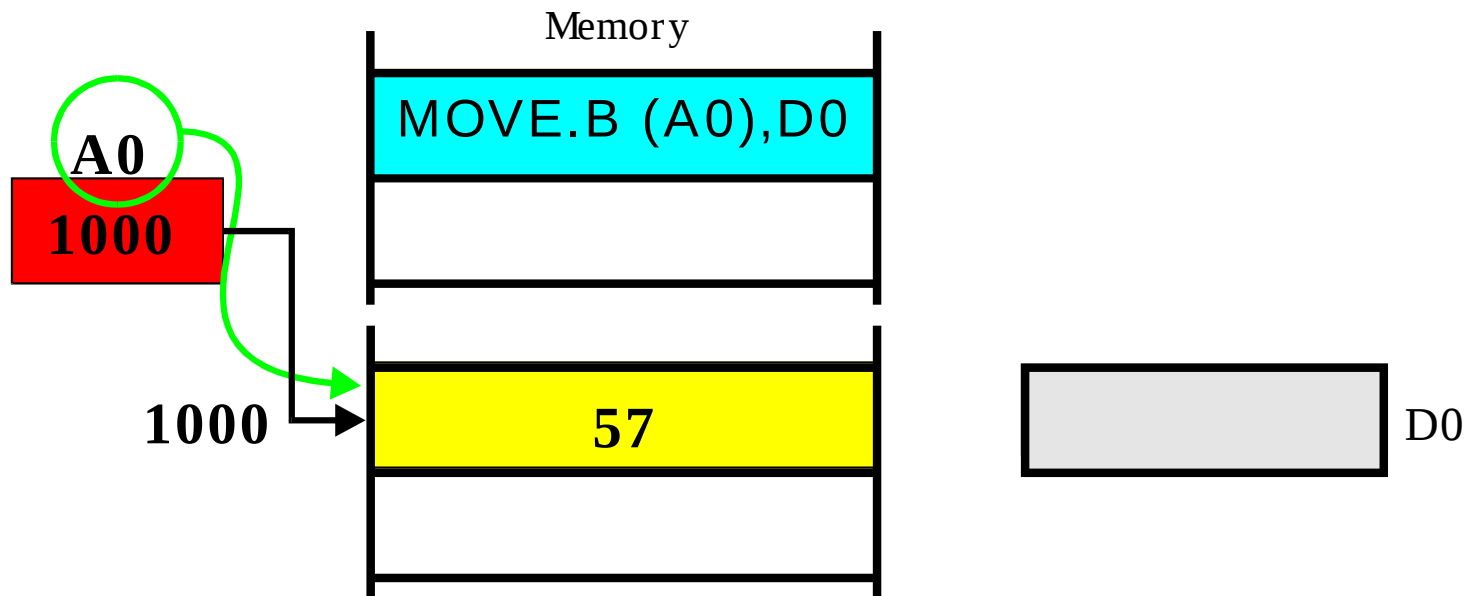
Instruktionen specificerar
source operand till (A0).

Instruktionen laddar D0
med innehållet i adress
positionen som A0 pekar
på.

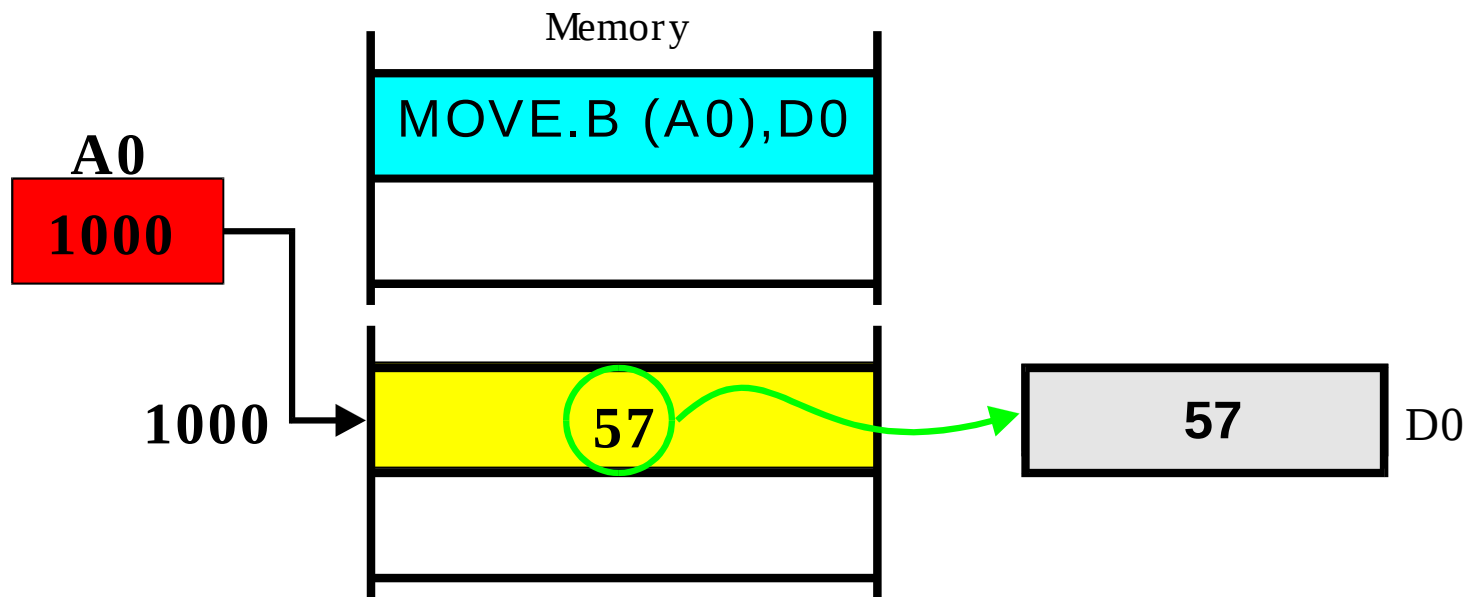




Adressen i register A0 specificerar
den adress där operanden lagras.



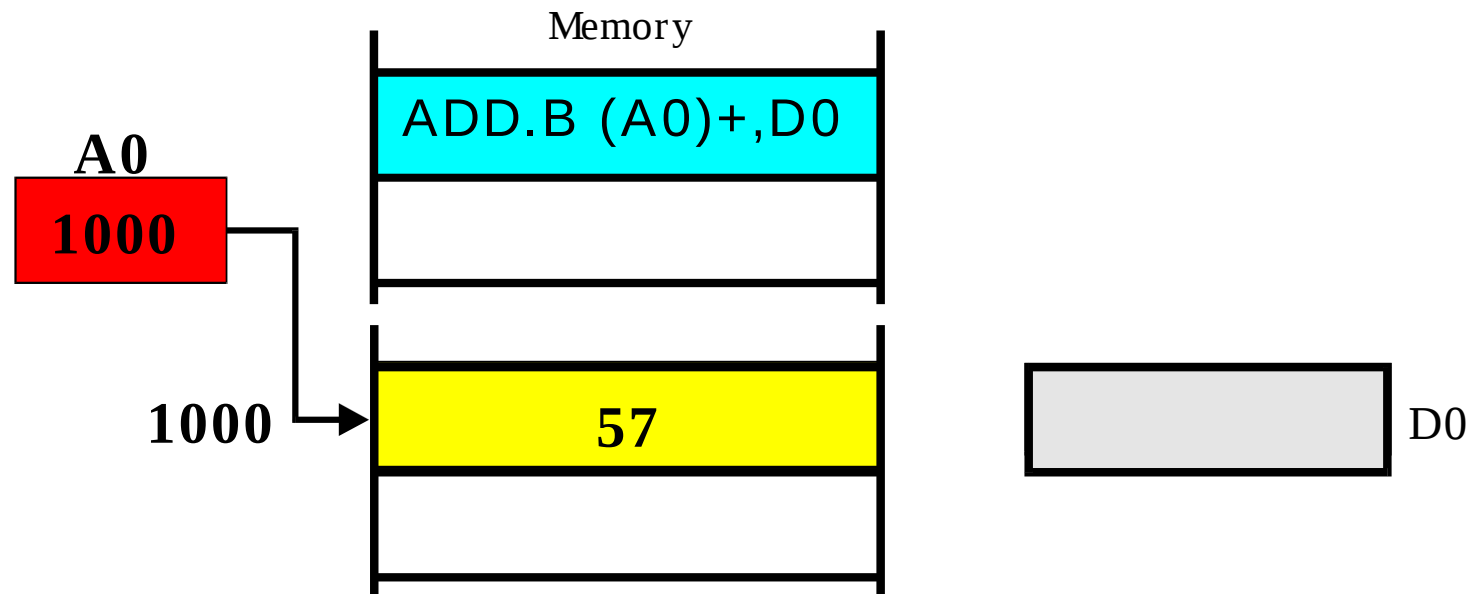
Adressregistret används för att läsa operanden från minnet.



Auto-increment och Auto-decrement

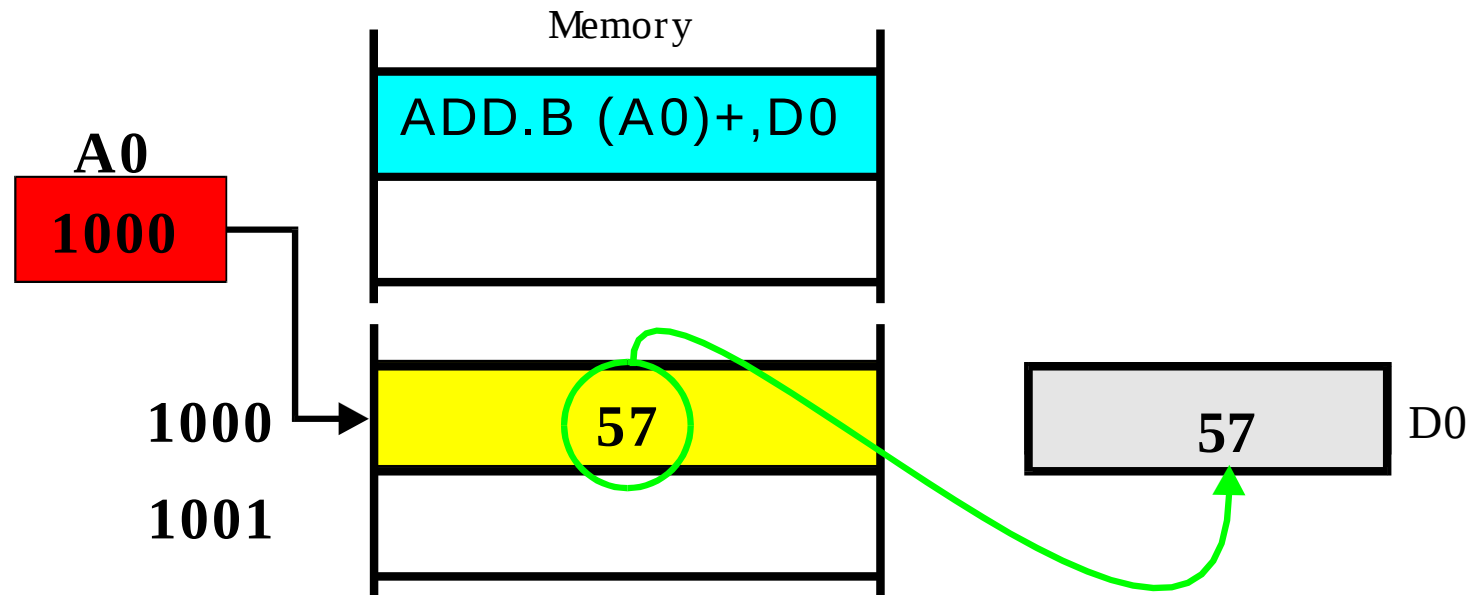
- Om operanden i en instruktion är specificerad som **(A0)+**, kommer A0 att ökas med en minnesposition **efter** det att operanden lästs/skrivits.
 - MOVE.B (A0)+, D0 Ökas A0 med 1
 - MOVE.W (A0)+, D0 Ökas A0 med 2
 - MOVE.L (A0)+, D0 Ökas A0 med 4
- Om operanden i en instruktion är specificerad som **-(A0)**, kommer A0 att minskas med en minnesposition **före** operanden läses/skrivs.

Post increment, #1



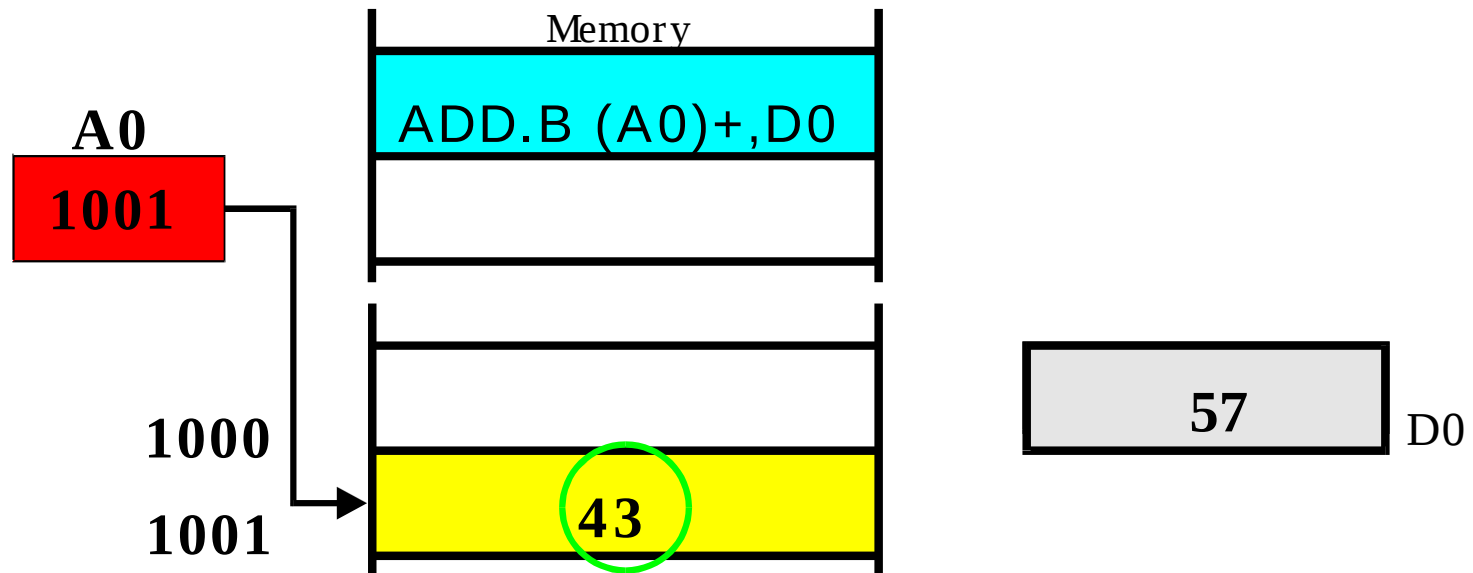
Adressregistret har värdet 1000
och pekar alltså på minnespositionen
med adressen 1000.

Post increment, #2



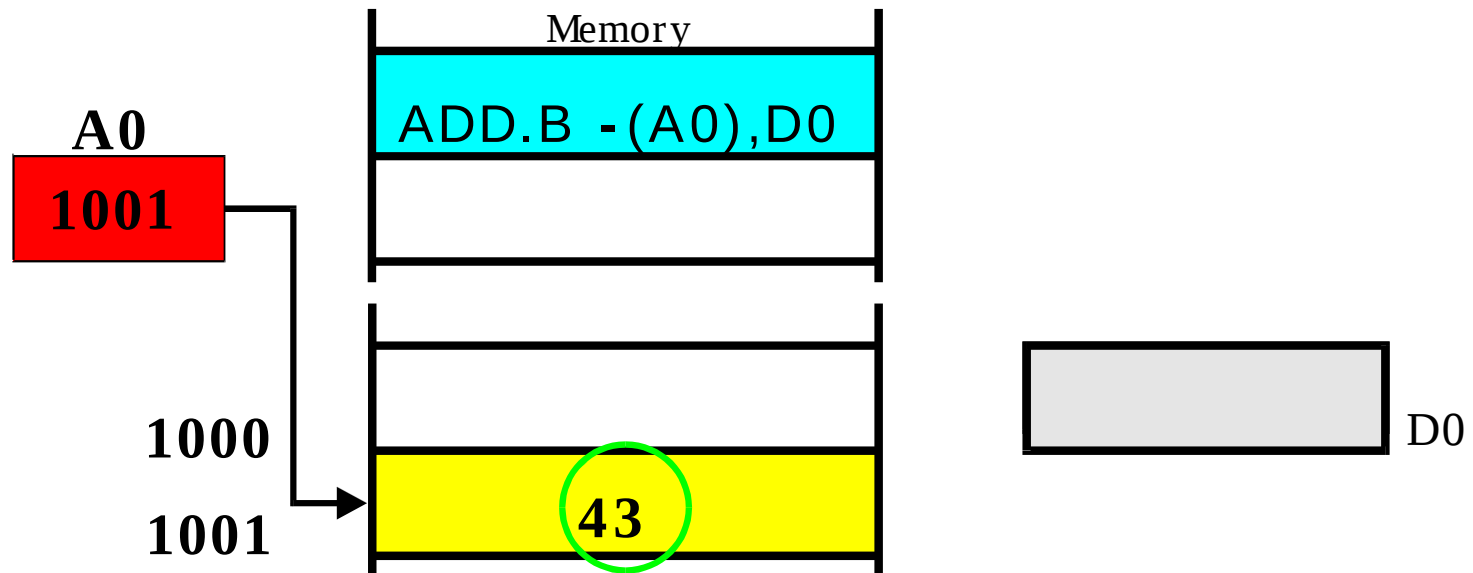
Adressen i register A0 används för att läsa minnesposition 1000. Innehållet i minnet adderas (dvs 57) till D0.

Post increment, #3



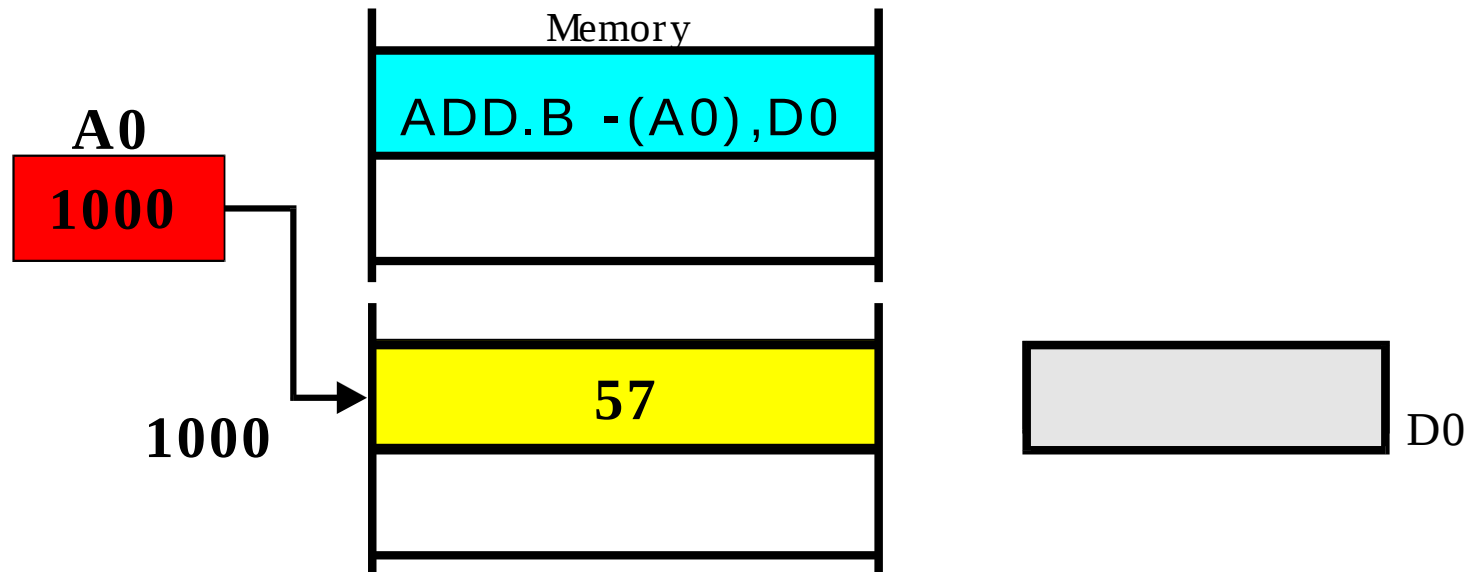
Efter de att instruktionen är exekverad kommer A0 att ökas med 1, och sedan peka på nästa minnesposition (1001)

Pre decrement, #1



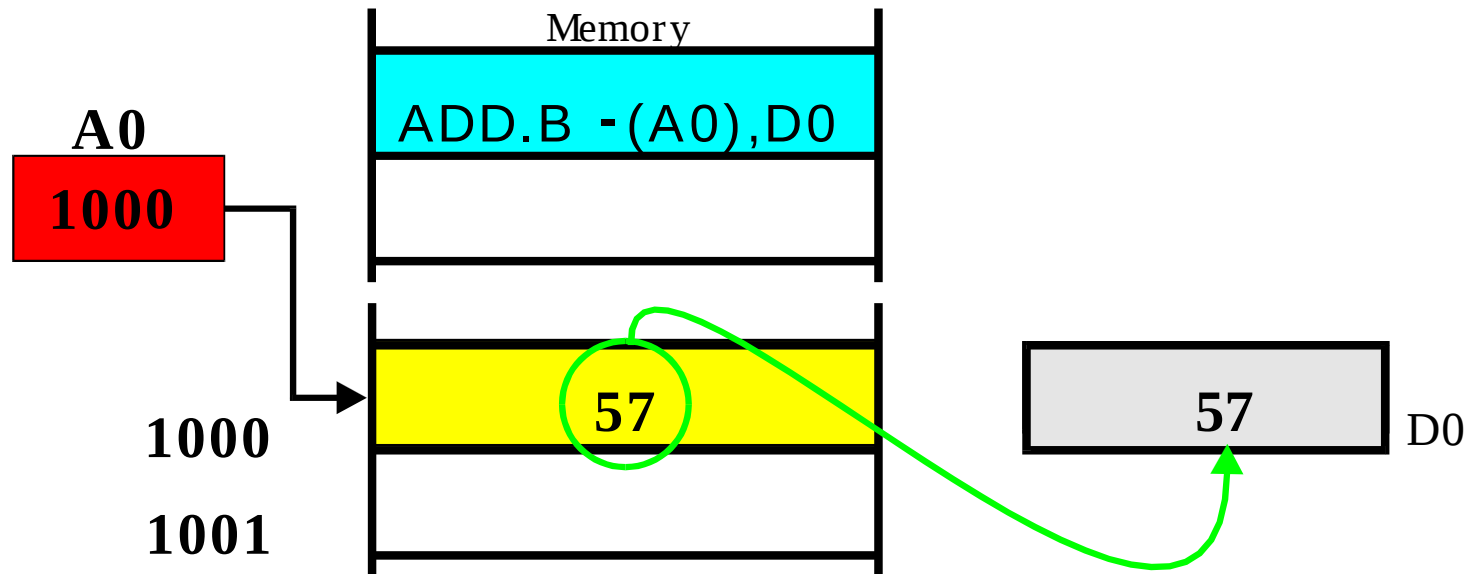
Adressregistret innehåller värdet 1001,
dvs det pekar på minnesposition 1001.

Pre decrement, #2



Före additionen utförs, minskas A0 med ett (byte).

Pre decrement, #3

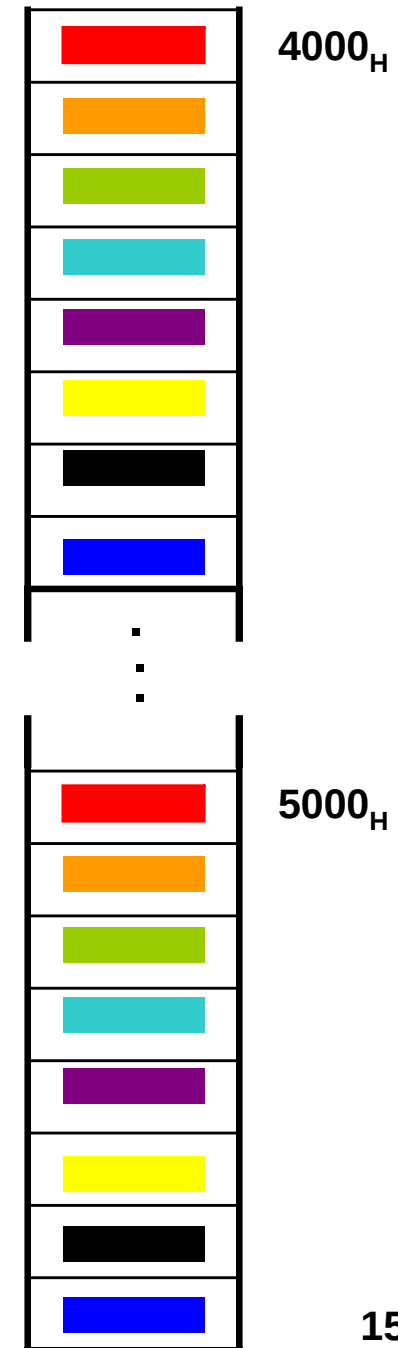


Adressen i register A0 används för att läsa minnesposition 1000. Innehållet i minnet adderas (dvs 57) till D0.

Ett exempel: 1

Ett block om 8 bytes skall kopieras
Från adressen 4000_H till 5000_H i
minnet.

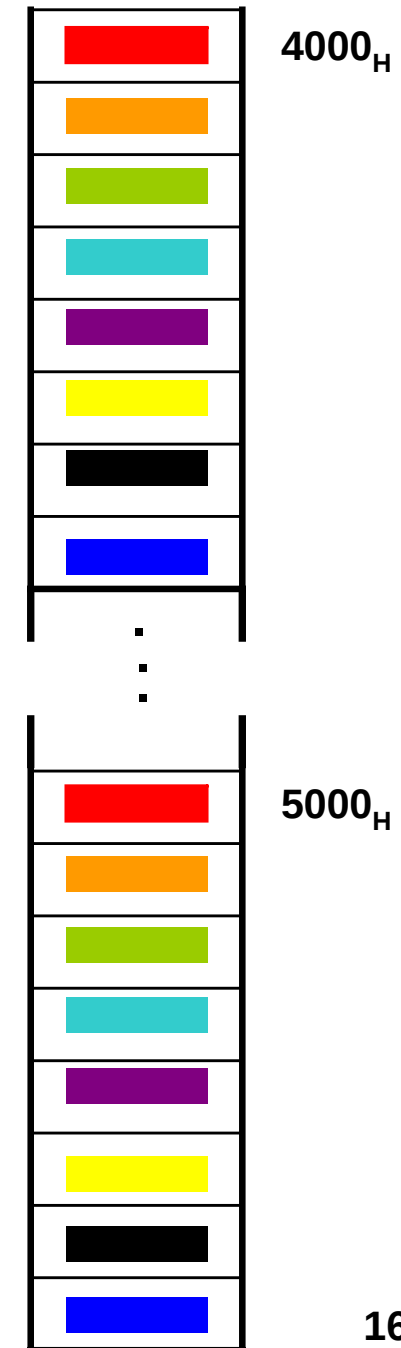
	<u>A0</u>	<u>A1</u>
MOVEA.L #\$4000, A0	4000	?
MOVEA.L #\$5000, A1	4000	5000
MOVE.W (A0)+,(A1)+	4002	5002
MOVE.W (A0)+,(A1)+	4004	5004
MOVE.W (A0)+,(A1)+	4006	5006
MOVE.W (A0)+,(A1)+	4008	5008



Ett exempel:2

Ett block om 8 bytes skall kopieras
Från adressen 4000_H till 5000_H i
minnet.

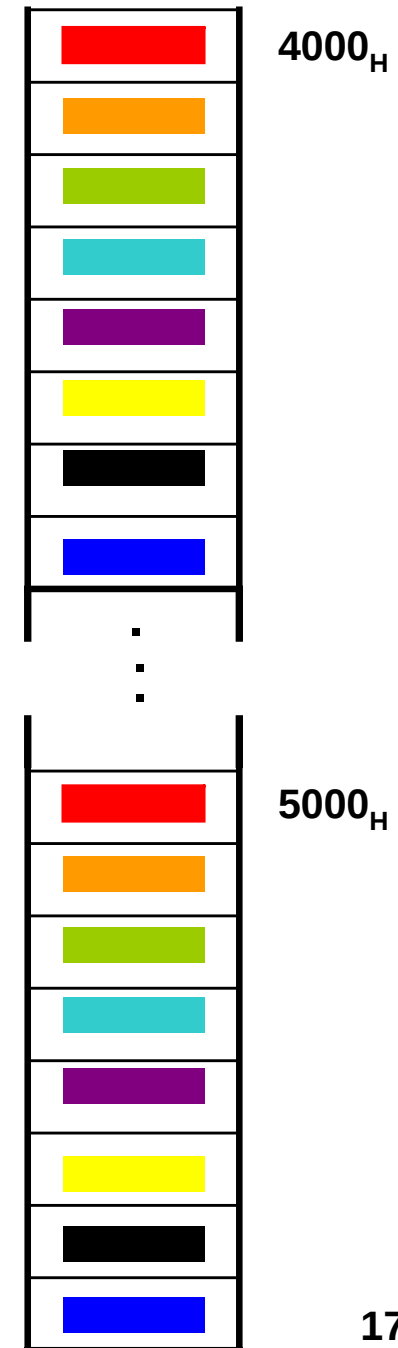
	<u>A0</u>	<u>A1</u>
MOVEA.L #\$4000, A0	4000	?
MOVEA.L #\$5000, A1	4000	5000
MOVE.L (A0)+,(A1)+	4004	5004
MOVE.L (A0)+,(A1)+	4008	5008



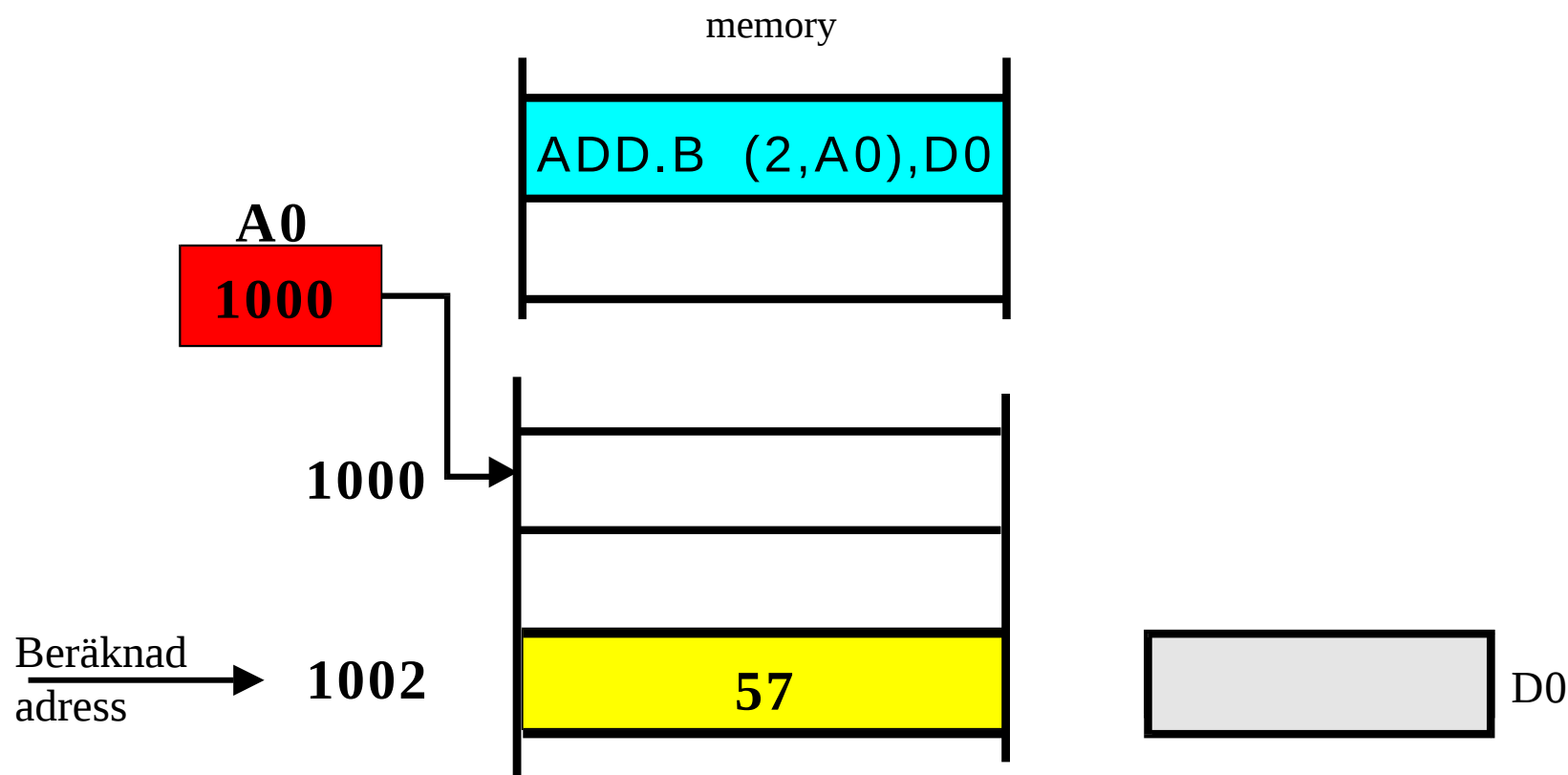
Ett exempel:3

Ett block om 8 bytes skall kopieras
Från adressen 4000_H till 5000_H i
minnet.

	<u>A0</u>	<u>A1</u>
MOVEA.L #\$4008, A0	4008	?
MOVEA.L #\$5008, A1	4008	5008
MOVE.W -(A0),-(A1)	4006	5006
MOVE.W -(A0),-(A1)	4004	5004
MOVE.W -(A0),-(A1)	4002	5002
MOVE.W -(A0),-(A1)	4000	5000

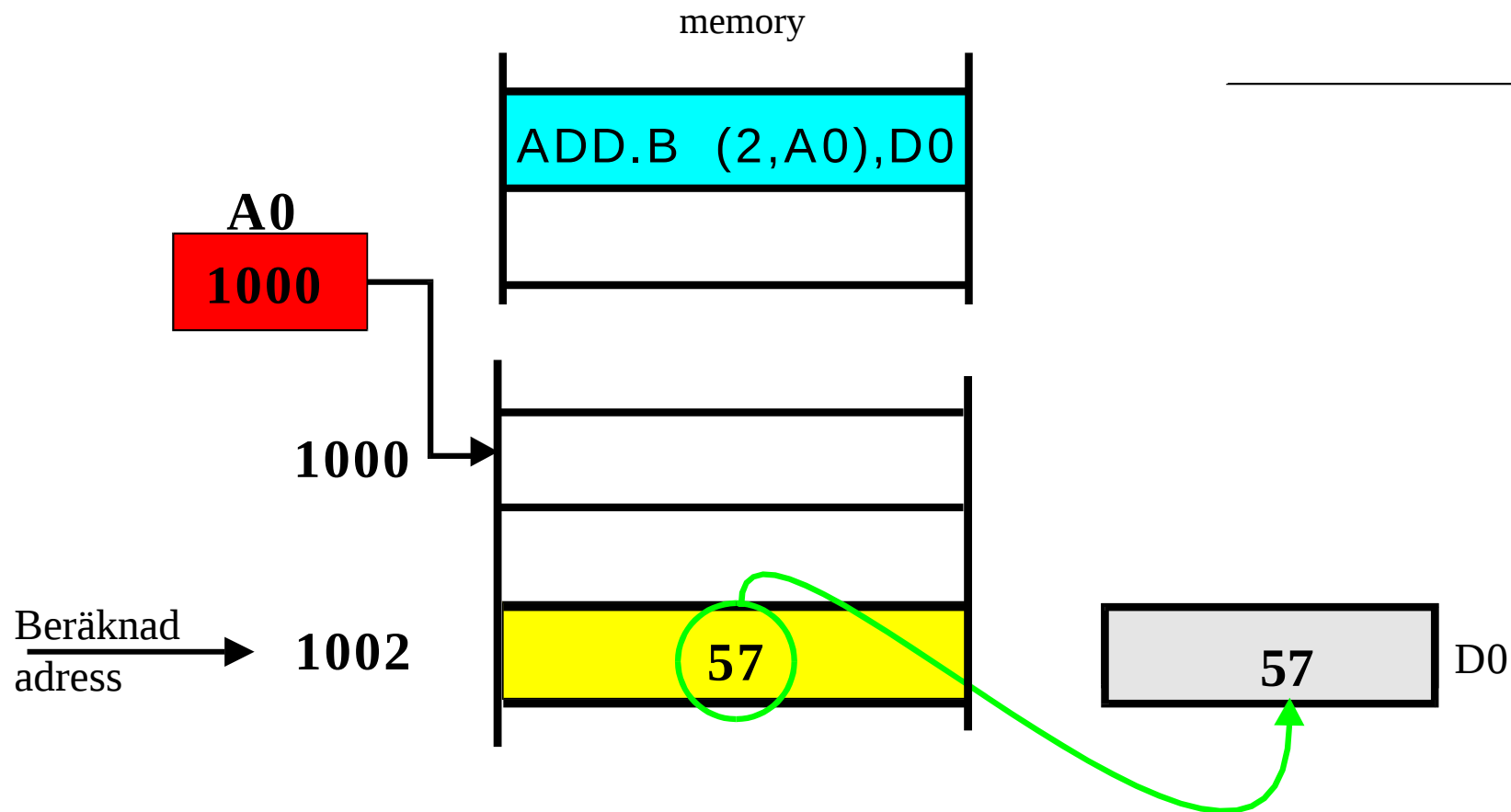


Indirekt adressering med förskjutning, #1



Till adressen i A0 adderas en förskjutning. Innehållet i minnet på den beräknade adressen skall adderas till D0.

Indirekt adressering med förskjutning, #2

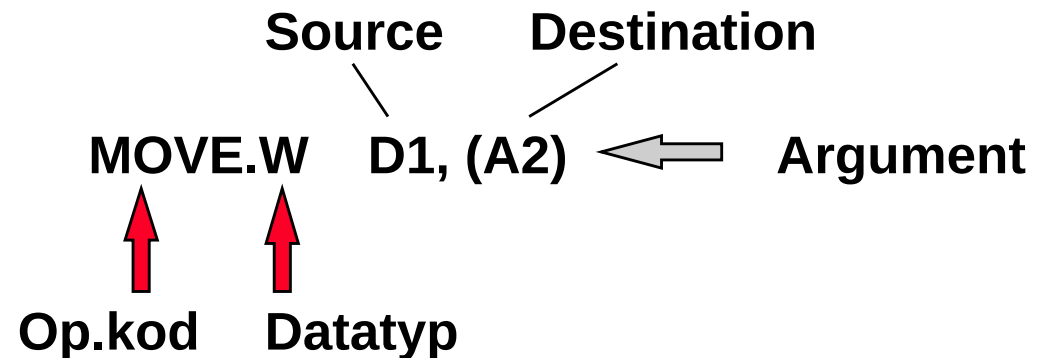


Assembler

- En assemblerinstruktion består av
 - Operationskod
 - Matematiska/ Logiska och dataflyttningsoperationer.
 - Datatyp
 - B = Byte, 8 bit.
 - W = Word, 16 bit.
 - L = Long word, 32 bit.

- Argument

- Dataregister
- Minnesposition
- Adressregister
- Osv..



Två typer av assembler instruktioner

- **Processorinstruktioner**
 - Används för skriva programmet
 - Exekverbara instruktioner
 - Specifik för processorn
- **Pre-processor instruktioner**
 - Används för att styra kompileringen
 - Icke exekverbara instruktioner
 - Specifik för assemblern

Pre-processorcommandon

- Dessa instruktioner exekveras inte utan ger direktiv till programmet (Assembler) som översätter koden från Assembler till Maskinkod.
- **EQU**
 - Sätter en identifierare till ett bestämt värde.
 - Ex.
 - IO EQU \$400700
- **ORG**
 - Sätter adressen för den aktuella positionen i programmet.
- **USE**
 - Används för att inkludera en fil

Preprocessorkommandon, forts.

- **Name** **DC.type** **Constant**
 - DC = Define Constant
 - **Name**, namnet på konstanten
 - **type**, bitbredd på operanderna [b,w,l]
 - **Constant**, värde som programmet initierar konstanten till
 - Assemblatorn reserverar plats i minnet och initierar detta till specificerade värden
 - Ex.
 - ORG \$12000
 - Kalle DC.L \$4F34
 - En_text DC.B 'Kalle',0

Preprocessor kommandon, forts.

- **Name** **DS.type** **Space**
 - DS = Define Storage
 - **Name**, namnet på variabeln
 - **type**, bitbredd på operanderna [b,w,l]
 - **Space**, antal positioner av type storlek ska reserveras
 - Assemblatorn reserverar plats i minnet specificerat antal minnespositioner
 - Ex.
 - ORG \$12000
 - Kalle DS.L 2 * Lagrar 2*4 byte (8 byte)
 - En_text DS.B 54 * Lagrar 54*1 byte (54 bytes)

Deklarationer

- **Konstant**

- `const int a = 3;`

- a DC.L 3

- **Variabel**

- `char kalle;`

- kalle DS. B 1

- **Array**

- `int arr[34];`

- arr DS.L 34

- **Textsträng**

- `char str = "Hello!";`

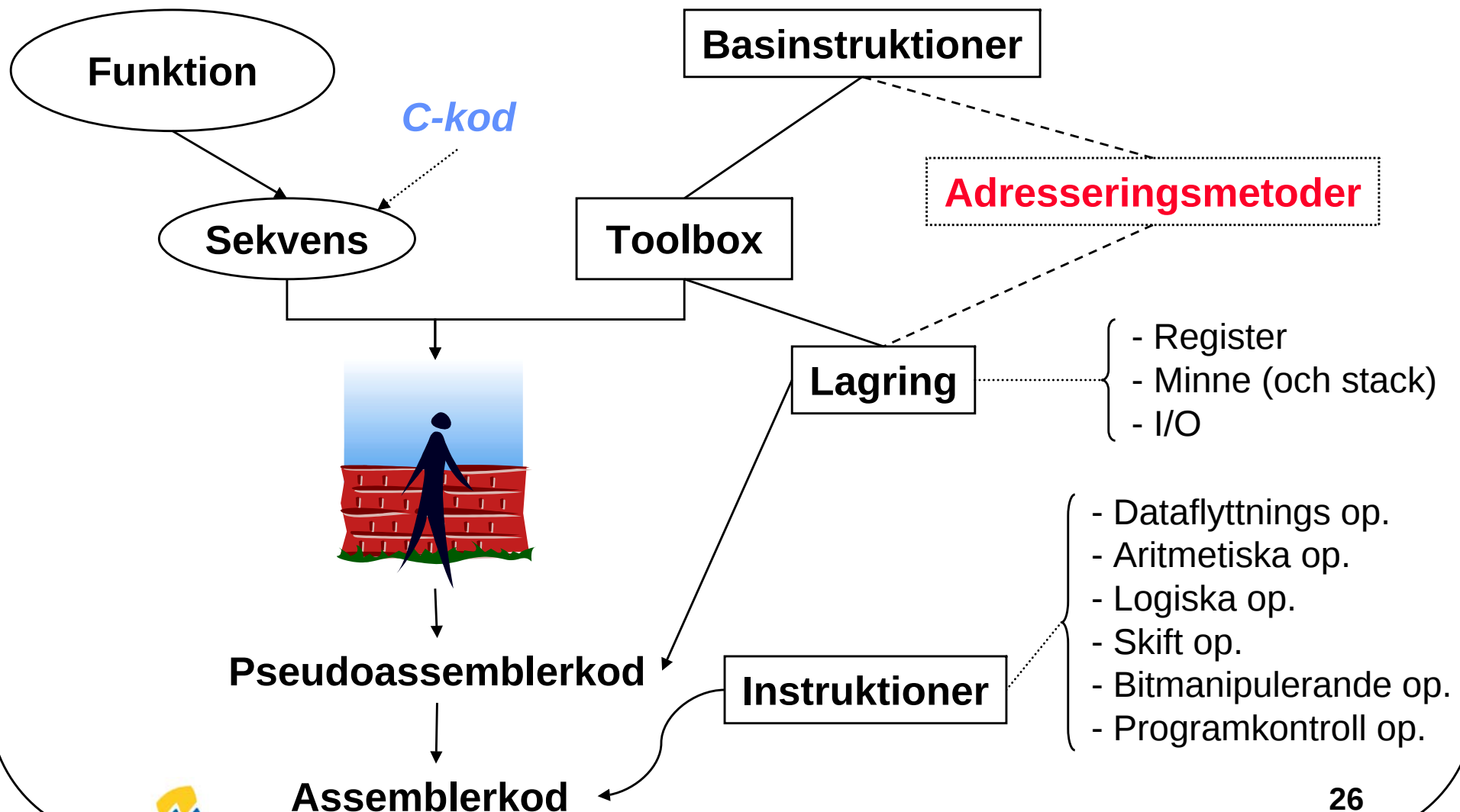
- str DC.B 'Hello!',0

- **Initierad vektor**

- `int vec={4,2,6,7,9};`

- vec DC.L 4,2,6,7,9

Assemblerprogrammering, metodik



Basinstruktioner

- **Pseudoassembler (13 instruktioner)**
 - **Dataflyttningsinstruktioner**
 - MOVE
 - **Aritmetikinstruktioner**
 - ADD, SUB, MUL, DIV
 - **Logikinstruktioner**
 - NOT, OR, AND, XOR
 - **Skiftinstruktioner**
 - SL, SR
 - **Hoppinstruktioner**
 - JMP, JIF

Exempel

- **Funktion: Beräkna $Y=a+b-c-3$**
- **Sekvens**
 - Som C kod
 - $y = a + b - c - 3;$
- **Att tänka på när man översätter C-koden till pseudoassembler**
 - En processor gör en sak i taget
 - Ladda **y** med **a**:s värde
 - Addera **b** till **y**
 - Subtrahera **c** från **y**
 - Subtrahera **3** från **y**

$y = a$

$y = a + b$

$y = a + b - c$

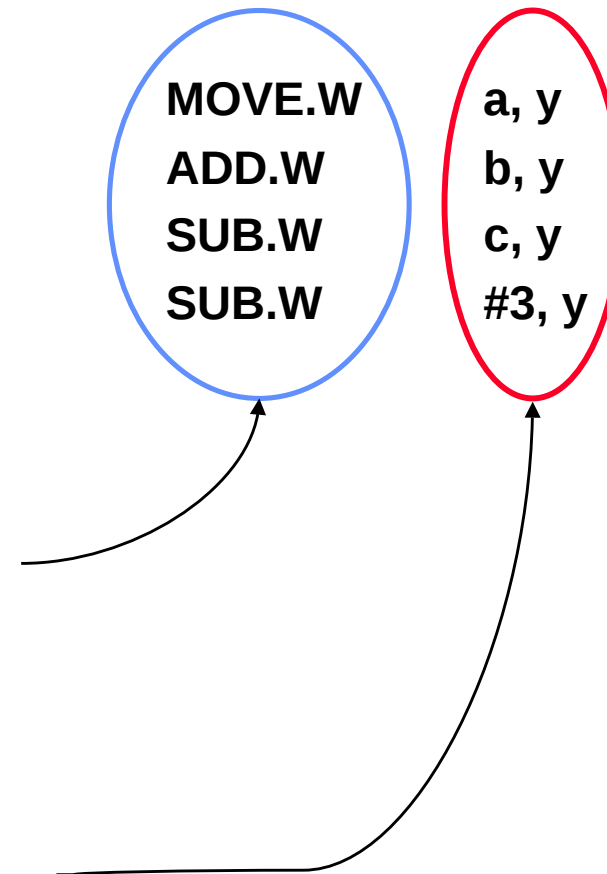
$y = a + b - c - 3$



Översättning till pseudoassembler

- Ladda **y** med **a**:s värde
- Addera **b** till **y**
- Subtrahera **c** från **y**
- Subtrahera **3** från **y**

- Nästa steg (Transformerering)
 - Transformera till implementerbar kod, dvs. 68000 assembler
 - Att tänka på:
 - Översätt varje instruktion till den riktiga 68000 instruktionen
 - Stämmer adresseringsmetoderna???
 - Om inte, gör variabelsubstitution



Transformerering till 68000-kod 1(2)

MOVE.W

a, y



ADD.W

b, y



SUB.W

c, y



SUBI.W

#3, y



- Substituera y med dataregister D0, eftersom att SUB och ADD kräver att någon av operatorerna ska vara dataregister.

Instruktionerna är konverterade till 68000-kod (OK)

Adresseringsmetoderna är inte giltiga enligt instruktionsmanualen => Variabelsubstitution

Transformerering, 2(2)

MOVE.W a, D1
ADD.W b, D1
SUB.W c, D1
SUBI.W #3, D1
MOVE.W D1, y

y är utbytt mot D1 under
programmets exekvering

För att resultatet
ska lagras i minnet,
kopieras D1 till y

Program i 68000-assembler som implementerar: $y = a+b-c-3$

MOVE.W a, D1
ADD.W b, D1
SUB.W c, D1
SUBI.W #3, D1
MOVE.W D1, y

Optimering

- **Minimera minnesaccesser, dvs. använd register när det lönar sig**

– **Tex:** Återanvändning av registervärden

ADD.W a,D0

ADD.W a,D1

ADD.W a,D2

18 bytes, 60 cykler

MOVE.W a,D3

ADD.W D3,D0

ADD.W D3,D1

ADD.W D3,D2

12 bytes, 28 cykler

- **Använd instruktioner som är snabba och kräver lite minne**

– **Tex:** Ekvivalenta instruktioner

MOVE.L #45, D0, kräver 12 klockcykler att exekvera (6 bytes)

MOVEQ.L #45, D0 kräver 4 klockcykler att exekvera (2 bytes)