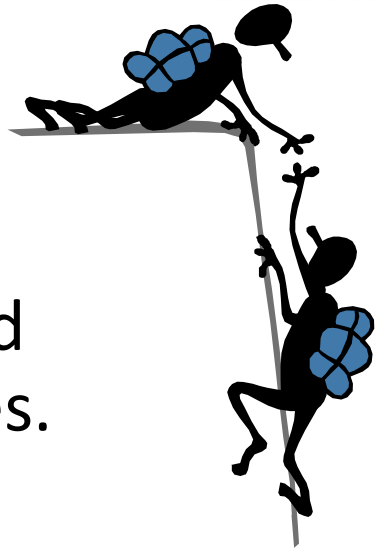
SEE 3223 Microprocessors

4: Addressing Modes

Muhammad Mun'im Ahmad Zabidi (munim@utm.my)

Addressing Modes



- Aims
  - To review 68000 data transfer instructions and applying the more advanced addressing modes.
- Intended Learning Outcomes
  - At the end of this module, students should be able to
    - Review simple addressing modes:
      - Register direct addressing mode (2 variations)
      - Immediate addressing mode
    - Understand:
      - Absolute addressing mode in detail
      - Address register direct addressing modes (2 variations)
      - Address register indirect addressing modes (5 variations)
      - Inherent addressing mode

68000 Addressing Modes

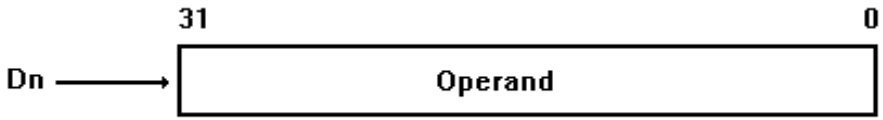
- The 68000 Addressing Modes are:
  - Register Direct Group
    - Data Register Direct*
    - Address Register Direct
  - Immediate*
  - Address Register Indirect Group
    - Address Register Indirect
    - Address Register Indirect with Postincrement
    - Address Register Indirect with Predecrement
    - Address Register Indirect with Displacement
    - Address Register Indirect with Index
  - Absolute Group
    - Absolute Short
    - Absolute Long
  - Program Counter Relative Group
    - Program Counter with Displacement
    - Program Counter with Index
  - Implicit

* Already covered

Data Register Direct

```
MOVE   D1,D2
```

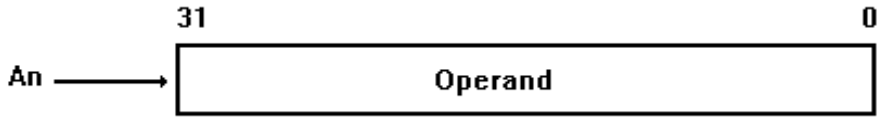
- The operand is found in the data register specified by the instruction.
- EA = Dn (data is found in a data register)
- Assembler Syntax: Dn



Address Register Direct

```
MOVE   A0,D2
```

- The operand is found in the address register specified by the instruction.
- EA = An (data is found in an address register)
- Assembler Syntax: An



Immediate Data

```
MOVE   #5,D2
```

- This addressing mode specifies the address of the operand in memory, the address follows the opcode. The address is specified high order byte first. The immediate data size is either Byte, Word or Long.
- Immediate value is assumed to be decimal unless indicated otherwise (ie by $ for hexadecimal or @ for octal).
- Uses: incrementing loop counters, working with immediate values.
- You know the actual value of the data
- EA = given
- Assembler Syntax: #xxx.SIZE

Immediate Addressing Mode Examples

•*Immediate*: an actual number X is provided.

Registers	
D2	XXXX XXXX
D3	XXXX XXXX
A0	0000 2000

Memory	
002000	1234
002002	5678
002004	ABCD

```
MOVE.B #12, D2
```

Registers	
D2	XXXX XX0C
D3	XXXX XXXX
A0	0000 2000

```
MOVE.W #$12, D2
```

Registers	
D2	XXXX 0012
D3	XXXX XXXX
A0	0000 2000

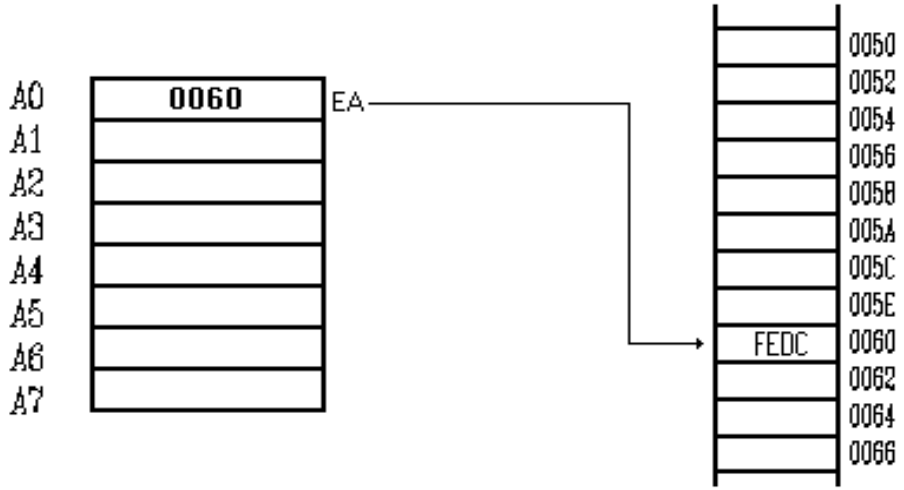
```
MOVE.L #12, D2
```

Registers	
D2	0000 000C
D3	XXXX XXXX
A0	0000 2000

Address Register Indirect (ARI) Addressing

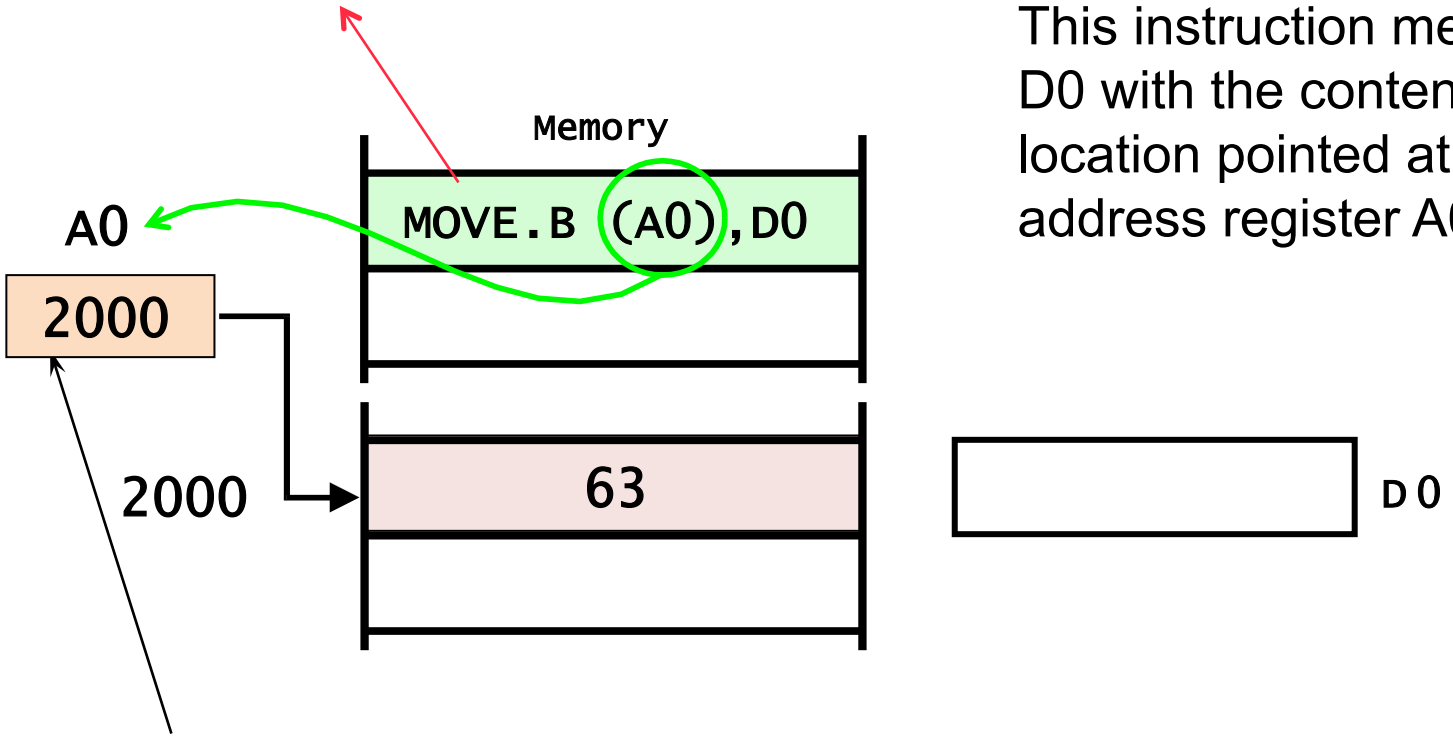
```
MOVE   (A0),D2
```

- This addressing mode specifies the operand in memory, the address of which is specified by one of the address registers.
- The operand is found in the address specified by an address register.
- Uses: repeated access to the same memory location
- EA = (An)
- Assembler Syntax:



Address Register Indirect Addressing

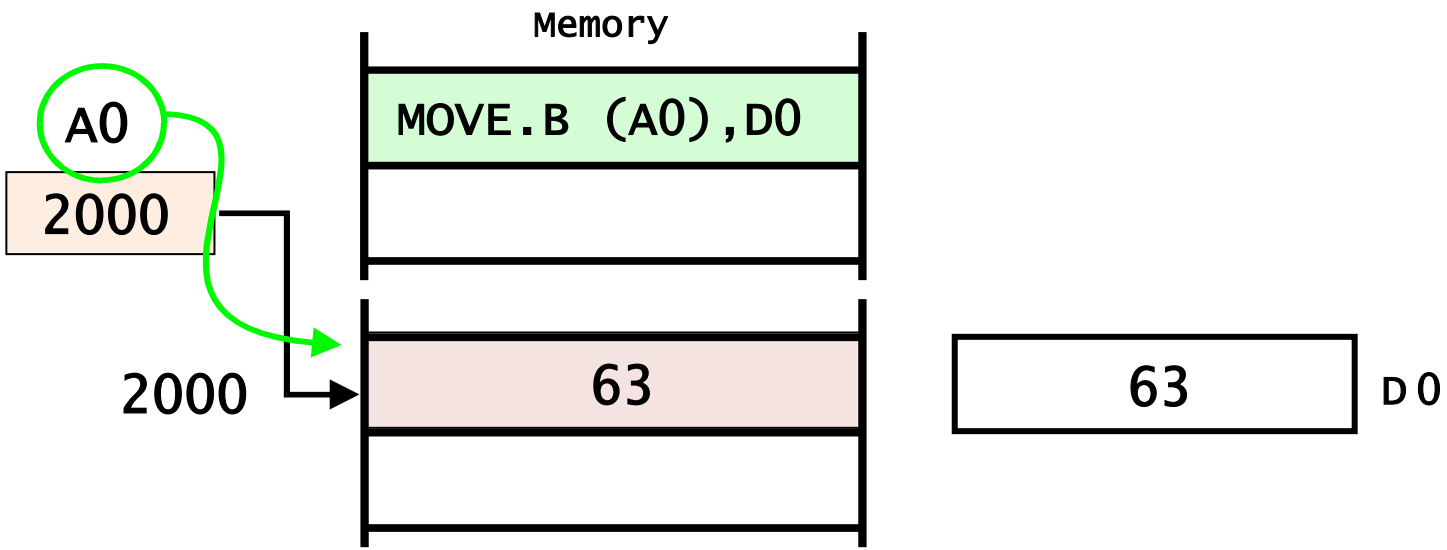
RTL Form: `[D0] ← [M([A0])]`



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

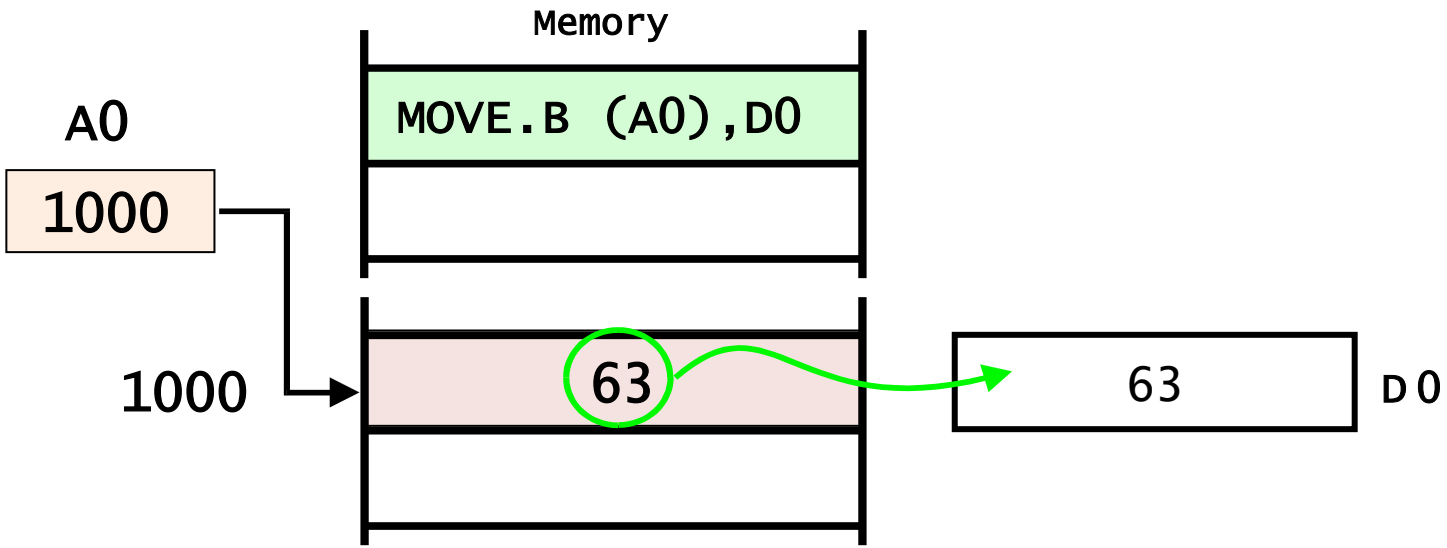
The address register in the instruction specifies an address register that holds the address of the operand

Address Register Indirect Addressing



The address register is used to access the operand in memory

Address Register Indirect Addressing



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

ARI Examples

•*Register Indirect*: accesses the contents of the memory location in the indicated register.

Registers	
D2	XXXX XXXX
D3	XXXX XXXX
A0	0000 2000

Memory	
002000	1234
002002	5678
002004	ABCD

```
MOVE.B (A0), D2
```

Registers	
D2	XXXX XX12
D3	XXXX XXXX
A0	0000 2000

```
MOVE.W (A0), D2
```

Registers	
D2	XXXX 1234
D3	XXXX XXXX
A0	0000 2000

```
MOVE.L (A0), D2
```

Registers	
D2	1234 5678
D3	XXXX XXXX
A0	0000 2000

ARI with Postincrement/ Predecrement

- In the 68000, the increment/decrement depends on the operand size
  - Suppose A0 = $00002000

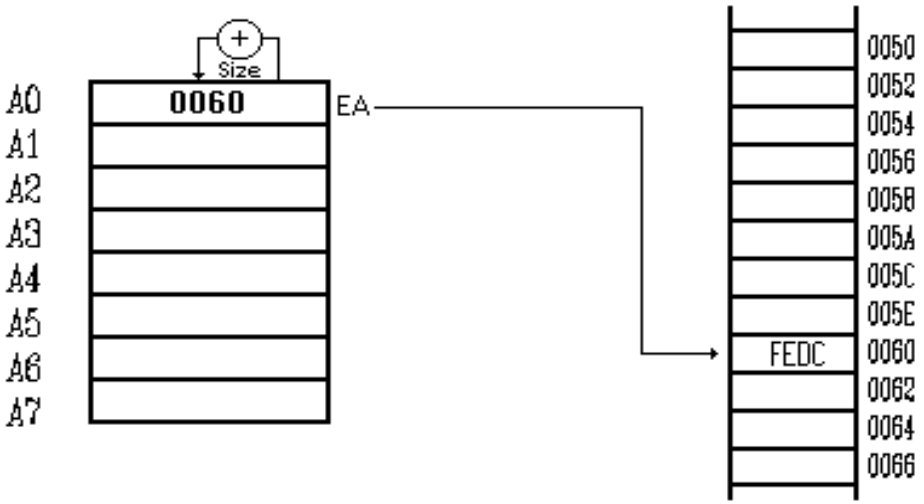
```
MOVE.B    (A0)+,D0   →   A0 = $00002001
MOVE.W    (A0)+,D0   →   A0 = $00002002
MOVE.L    (A0)+,D0   →   A0 = $00002004
```

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

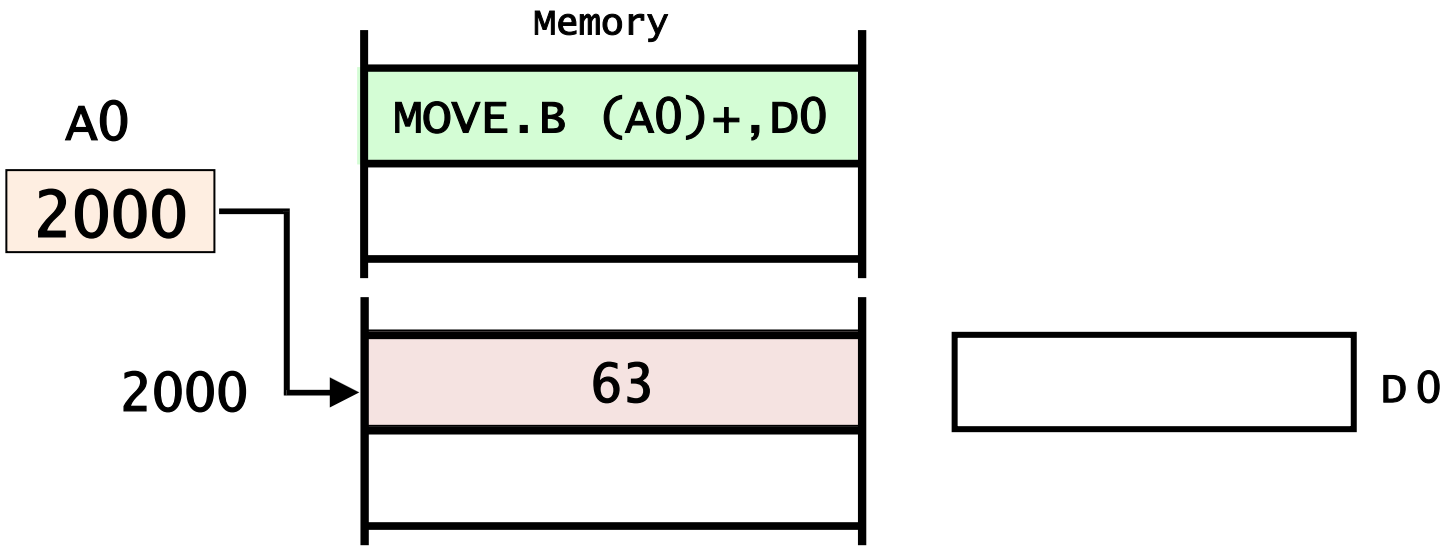
ARI with Postincrement

```
MOVE    (A0)+,D2
```

- This addressing mode specifies the operand in memory, the address of which is specified by one of the address registers. After the operand is used, the value in the address register is incremented according to the size of the operand.
  - +1 byte
  - +2 word
  - +4 long word
- The operand is found in the address specifie
- Uses: moving through an array, popping fror
- EA = (An); An incremented after use
- Assembler Syntax: (An)+

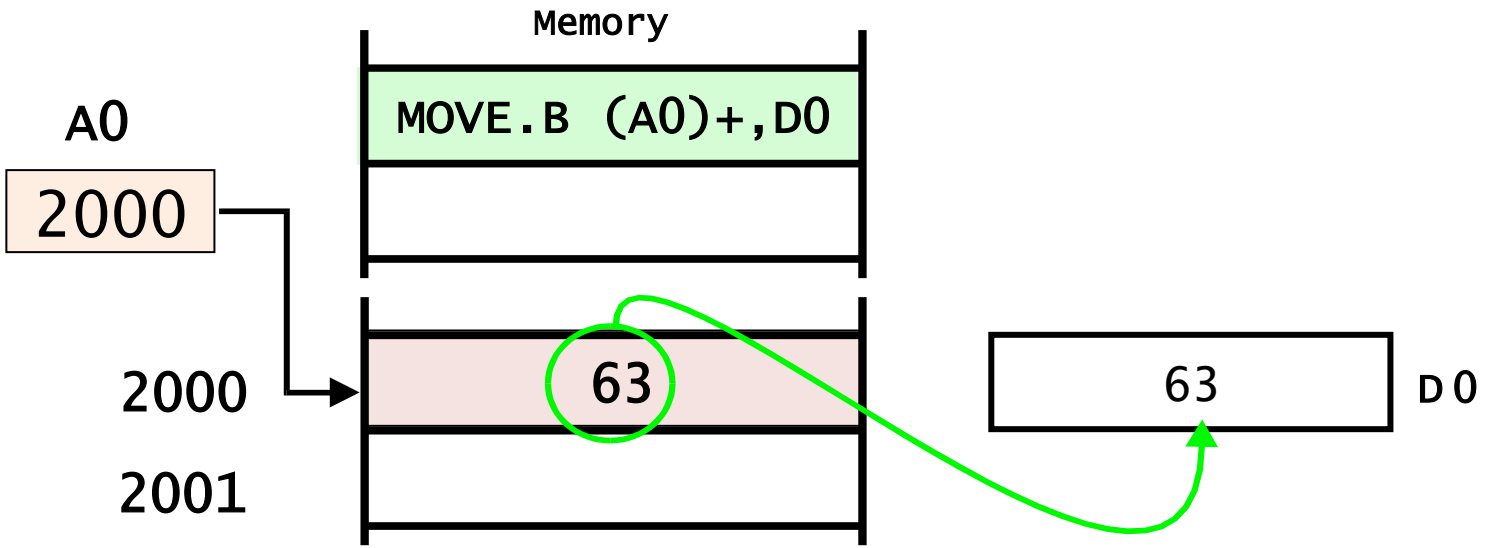


ARI with Postincrement



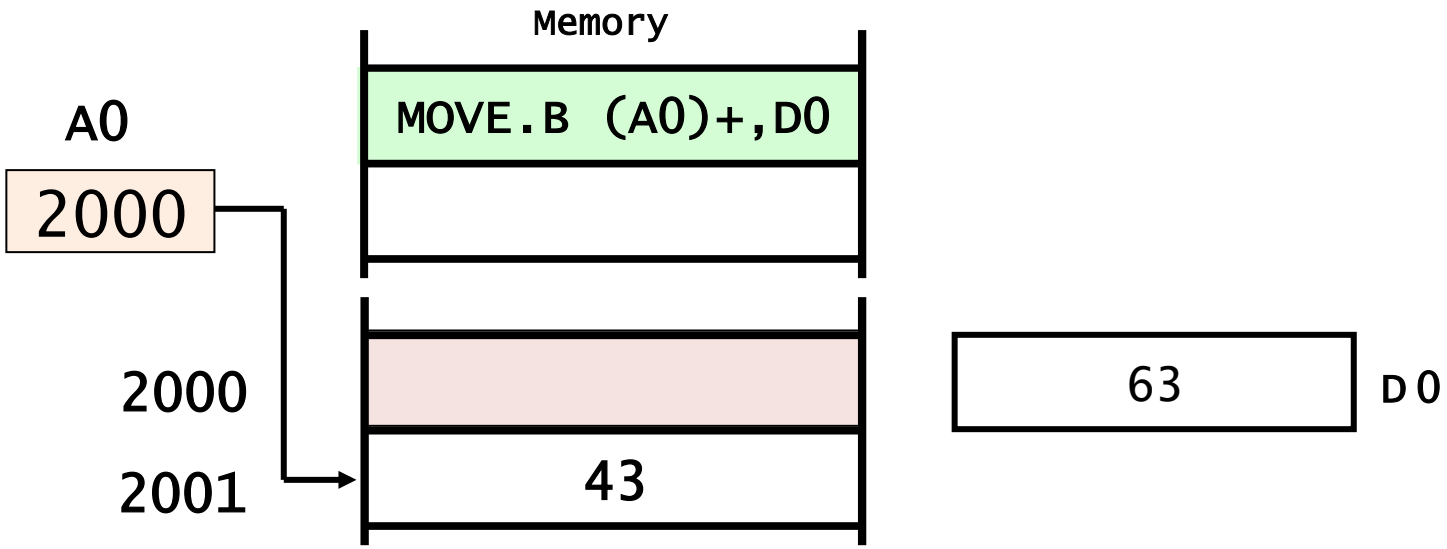
The address register contains 2000
and points at location 2000

ARI with Postincrement



Address register A0 is used to access memory Location 2000 and the contents of this location (i.e., 63) are added to D0

ARI with Postincrement



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

ARI with Postincrement Examples

•*Post-increment:* Operand is accessed indirectly, then address register is incremented.

Registers	
D2	XXXX XXXX
D3	XXXX XXXX
A0	0000 2000

Memory	
002000	1234
002002	5678
002004	ABCD

```
MOVE.B (A0)+, D2
```

Registers	
D2	XXXX XX12
D3	XXXX XXXX
A0	0000 2001

```
MOVE.W (A0)+, D2
```

Registers	
D2	XXXX 1234
D3	XXXX XXXX
A0	0000 2002

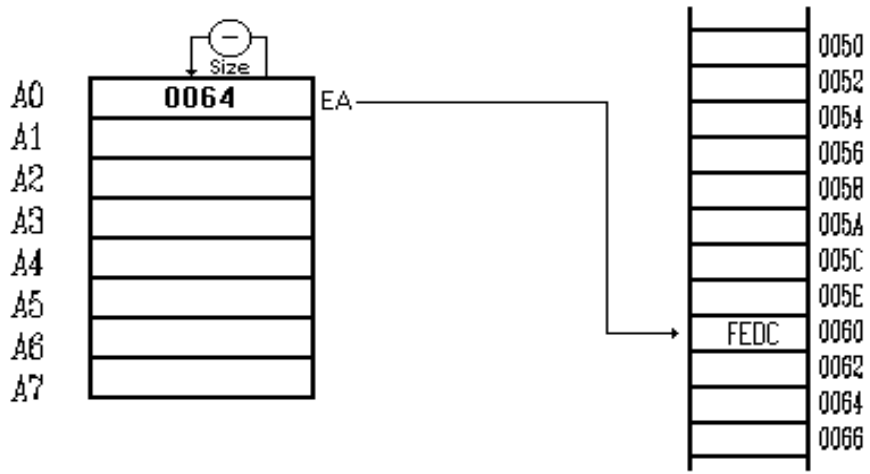
```
MOVE.L (A0)+, D2
```

Registers	
D2	1234 5678
D3	XXXX XXXX
A0	0000 2004

ARI with Predecrement

MOVE -(A0),D2

- This addressing mode specifies the operand in memory, the address of which is specified by one of the address registers. Before the operand is used, the value in the address register is decremented according to the size of the operand.
  - -1 byte
  - -2 word
  - -4 long word
- The operand is found in the address specified by an address register.
- Uses: moving through an array, pushing onto stack
- EA = (An) – SIZE; An decremented before use
- Assembler Syntax: -(An)



ARI with Predecrement Examples

•*Pre-decrement:*
Address register is decremented, then operand is accessed indirectly.

Registers	
D2	XXXX XXXX
D3	XXXX XXXX
A0	0000 2004

Memory	
002000	1234
002002	5678
002004	ABCD

`MOVE.B -(A0), D2`

Registers	
D2	XXXX XX78
D3	XXXX XXXX
A0	0000 2003

`MOVE.W -(A0), D2`

Registers	
D2	XXXX 5678
D3	XXXX XXXX
A0	0000 2002

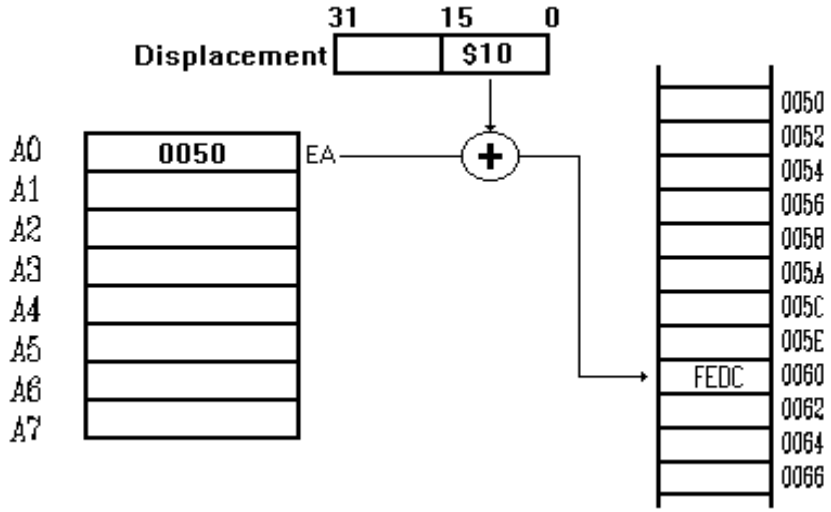
`MOVE.L -(A0), D2`

Registers	
D2	1234 5678
D3	XXXX XXXX
A0	0000 2000

ARI with Displacement

```
MOVE   16(A0),D2
```

- An 16-bit displacement value is added to the memory address in the indicated register to form the effective address, then the contents of the effective address are accessed
- EA = (An) + d_{16}
- Assembler Syntax: d_{16}(An)



ARI with Displacement Examples

ARI with Displacement: An index value is added to the memory address to form the effective address.

Registers	
D2	XXXX XXXX
D3	XXXX XXXX
A0	0000 2002

Memory	
002000	1234
002002	5678
002004	ABCD

MOVE.W 2(A0), D2

Registers	
D2	XXXX ABCD
D3	XXXX XXXX
A0	0000 2002

MOVE.W -2(A0), D2

Registers	
D2	XXXX 1234
D3	XXXX XXXX
A0	0000 2002

MOVE.L -2(A0), D2

Registers	
D2	1234 5678
D3	XXXX XXXX
A0	0000 2002

ARI with Displacement Examples

Memory

Address	Value
002000	95
002002	89
002004	83

```
MOVE.L   #$002000,A0
CLR.L    D1
ADD.W    (A0),D1
ADD.W    2(A0),D1
ADD.W    4(A0),D1
DIV.W    #3, D1
```

```
struct Student {
        int grade1;
        int grade2;
        int grade3;
};

struct Student Ali, Kumar;

Total_Ali = Ali.grade1 +
            Ali.grade2 +
            Ali.grade3;

Avg_Ali = Total_Joe / 3;
```

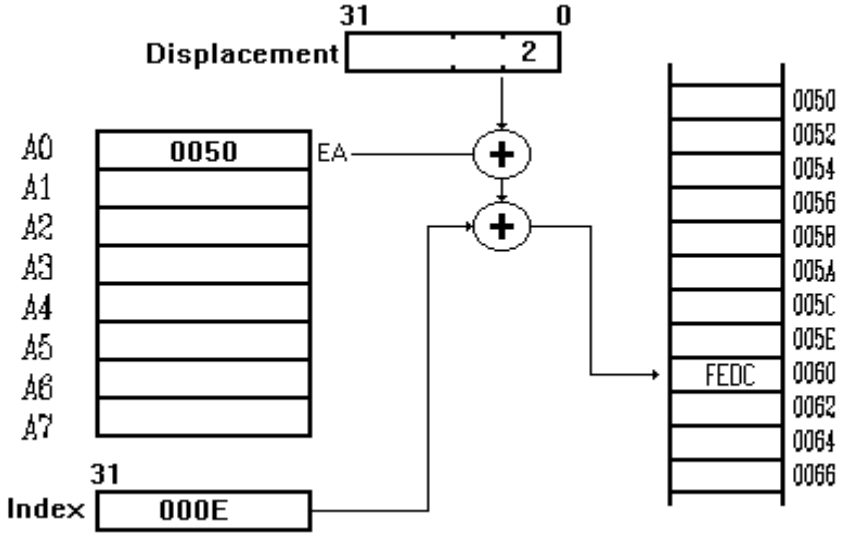
ARI with Index

```
MOVE   2(A0,D0),D2
```

- This addressing mode specifies the operand in memory, the address of which is specified by one of the address registers plus the value in an index register, plus the sign extended 8 bit displacement specified as part of the instruction.

$$EA = (An) + (Xn) + d_8$$

- Assembler Syntax: d_8(An,Xn.SIZE)



Absolute 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.
- Involves memory access
  - Store operation: data from processor to memory
  - Load operation: data from memory to processor
- Uses: moving stored variables from memory into registers for processing, storing results back to memory.
- You know the actual address ($001020) of the data, so you need to get it from there.

Absolute Long

```
MOVE   $001020,D2
```

- This addressing mode specifies the address of the operand in memory, the address of which is specified by two extension words which follow the opcode. The address is specified high order byte first.
- EA = given
- Assembler Syntax: xxx.L

Absolute Long Examples

•*Absolute Long:*
accesses the contents of the indicated memory location.

Registers	
D2	XXXX XXXX
D3	XXXX XXXX
A0	0000 2000

Memory	
002000	1234
002002	5678
002004	ABCD

```
MOVE.W $002000, D2
MOVE.B $002000, D2
MOVE.L $002000, D2
```

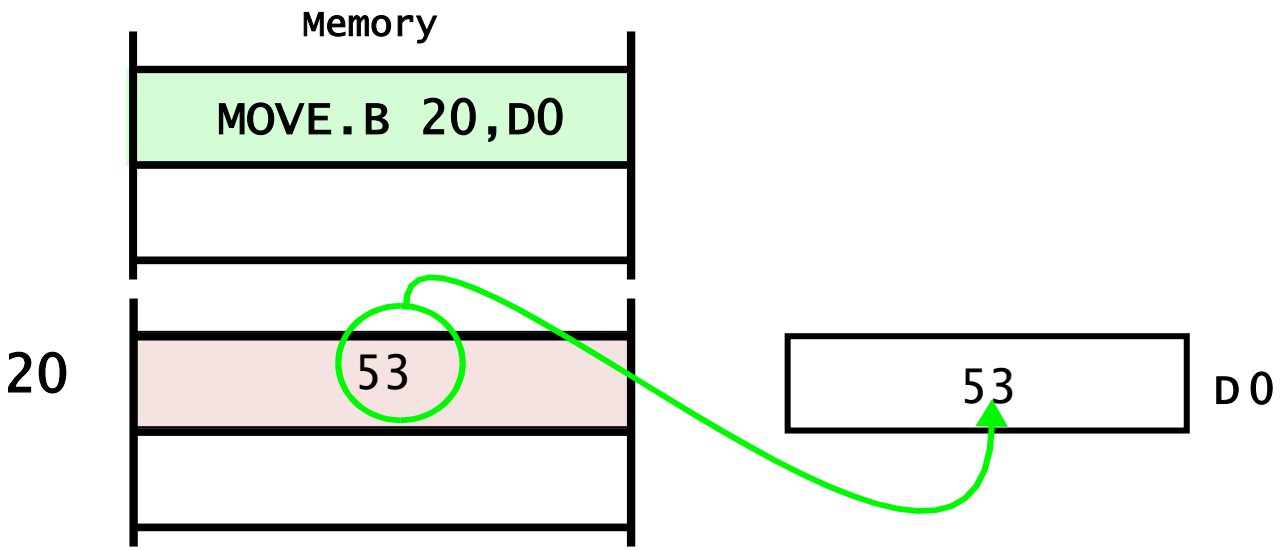
Registers	
D2	XXXX 1234
D3	XXXX XXXX
A0	0000 2000

Registers	
D2	XXXX XX12
D3	XXXX XXXX
A0	0000 2000

Registers	
D2	1234 5678
D3	XXXX XXXX
A0	0000 2000

Absolute Addressing

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



Absolute Short

```
MOVE   $1020,D2
```

- This addressing mode specifies the address of the operand in memory, the address of which is specified by one extension word which follow the opcode. The 16 bit address is signed extended to 32 bits before being used.

EA = given

- Assembler Syntax: xxx.W

- Uses: similar to absolute long, but saves a word in the instruction
- Assembler will determine whether the memory address is suitable for absolute short. Programmer normally don't have to know which mode (short/long) is used, but we can tell by the address used by the instruction.
- Limited to address $000000 to $007FFF, and $FF8000 to $FFFFFF.

Program Counter with Displacement

- This addressing mode permits memory to be accessed relative to the current value of the Program Counter. The major use is for jumps in position independent code (PIC), and reading constants in code segments.

$$EA = (PC) + d_{16}$$

- Assembler Syntax: $d_{16}(PC)$

```
* Simple example of PC addressing

        MOVE.B TABLE(PC),D2
               …
TABLE   DC.B   Value1
        DC.B   Value2
```

Program Counter with Index

- This addressing mode extends the program counter relative mode to include an index and offset value. The effective address of the operand is the sum of the extension word, a sign extended 8-bit displacement integer, and the contents of an index register. This effectively handles lists or tables.

$$EA = (PC) + (Xn) + d_8$$

- Assembler Syntax: d_8(PC,Xn.SIZE)

```
* Simple example of PC addressing

        MOVE.W   #2,D0
        MOVE.B   TABLE(PC,D0),D2
                 ...
TABLE   DC.B     Value1
        DC.B     Value2
```

Inherent

- CCR = Condition Code Register (8bit)
  - to CCR
    - ANDI
    - EORI
    - MOVE
    - ORI
- SR = Status Register (16bit)
  - to SR
    - ANDI
    - EORI
    - MOVE
    - ORI
  - from SR
    - MOVE

Memory Usage

- How many bytes for a 68000 instruction?
  - 68000 has 16-bit word size, 24-bit address size
  - Thus, one word fills 2 bytes, an address specification fills 2 words
- Depending on instruction, an **instruction word** is followed by **optional extension words** for each operand

– Data or address register	none
– Absolute address	2 ext. words
– Immediate value	1 or 2 ext. words
– Indirect register address	none
– Autoincrement/autodecrement	none
– Indexed/relative	1 ext. word
– Absolute/immediate	1 ext. word for byte/word, 2 ext. words for long

Fancy Uses of Addressing Modes

```
*
* Block copy: copy 10 words from $2000-$2009 to $3000-$3009
*
        ORG     $1000
        LEA     SRCBLK,A0
        LEA     DSTBLK,A1
        MOVE.W  #10,D0
LOOP    MOVE.B  (A0)+,(A1)+
        SUB.W   #1,D0
        BNE     LOOP
        STOP    #$2000
*
* Data section
*
        ORG     $2000
SRCBLK  DC.W    1,4,9,16,25,36,49,64,81,100
DSTBLK  EQU     $3000
        END     $1000
```

Fancy Uses of Addressing Modes

```
*
* String reverse: reverse a string and copy new string
*
        ORG     $1000
        LEA     STRING1,A0
        LEA     STRING2,A1
        MOVE.W  #LENGTH,D0
LOOP    MOVE.B  (A0)+,-(A1)
        SUB.W   #1,D0
        BNE     LOOP
        STOP    #$2000
*
* Data section
*
STRING1 DC.B    '68000 IS FUN'
LENGTH  EQU     *-STRING1
STRING2 EQU     *+LENGTH
        END     $1000
```

Programming Example 1

- Exchange words in $100 and $102

Solution 1:

```
MOVE.L  $100,D0 ; load example
SWAP    D0      ; Swap top word with bottom word
MOVE.L  D0,$100 ; store example
```

Solution 2:

```
MOVE.W  $102,$104           ; memory to memory
MOVE.W  $100,$102
MOVE.W  $104,$102
```

Programming Example 2

- Clear high word of D0

Solution 1:

```
SWAP    D0
CLR     D0
SWAP    D0
```

Solution 2:

```
MOVE.L  D0,$100
CLR.W   $100
MOVE.L  $100,D0
```

Summary - "Easy" Addressing Modes

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

```
ADD.B D1,D0
```

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

```
ADD.B #24,D0
```

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

```
ADD.B 1000,D0
```

Summary - ARI Addressing Modes

- Address Register Indirect (ARI):

```
ADD.B (A0),D0      ; No change to A0
```

- ARI with Postincrement:

```
ADD.B (A0)+,D0     A0 is incremented after use
```

- ARI with Predecrement

```
MOVE.L -(A0),D3    A0 is first decremented by 4
```

- ARI with Displacement

```
MOVE.L -2(A0),D3   ; No change to A0
```

- ARI with Index

```
MOVE.L -2(A0,D0),D3       ; No change to A0
```

Summary - Other Addressing Modes

- Program Counter Relative Addressing
  - Program Counter With Displacement

    ```
    MOVE.W   2(PC),D0
    ```
  - Program Counter With Index

    ```
    MOVE.W   2(PC,D3),D0
    ```
- **PC can be used only for SOURCE OPERANDS**