absolute short
(mode=111, register=000)

As shown above this addressing mode LOOKS like immediate but is NOT because no # precedes the label.

general form	ADD	LABEL,D3	←	Source is absolute (an address specified by a symbol); destination is data register direct

The source is a memory location. Specifically, LABEL will have an assigned value in the symbol table and this value will be used as the address of the source data. Unlike the immediate mode, this mode uses the standard ADD instruction format.

Recall the general form of the ADD instruction:

15	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
Opcode				Register			Op-mode			Effective Address Mode			Effective Address Register		

The assembled form for absolute short source addressing is:

15	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	1	0	0	1	1	1	1	0	0	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

where xxxxxxxxxxxxxxxx is be a 16-bit extension word containing the value of label and is used as the address of the data to be used in the ADD. Compared to immediate mode, this is the address of the data. Immediate mode would put the data in the extension word.

IMPORTANT

general form ADD #LABEL,D3 ← The source is NOT absolute, it is IMMEDIATE.

THIS IS THE #1 STUDENT ERROR.

The correct interpretation of this instruction is:

<table>
<tr><td>15</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">0</td><td colspan="2" rowspan="2">Size</td><td colspan="6">Effective Address</td></tr>
<tr><td colspan="3">Mode</td><td colspan="3">Register</td></tr>
<tr><td colspan="8">Word Data (16 bits)</td><td colspan="8">Byte Data (8 bits)</td></tr>
<tr><td colspan="16">Long Data (32 bits, including previous word)</td></tr>
</table>

where Word Data or Long Data is the value of LABEL interpreted as a constant. For example, if LABEL had the value $7EFF the above instruction would add the value $7EFF to the contents of D3. IT WOULD NOT ADD THE CONTENTS OF MEMORY ADDRESS $7EFF to D3.

ADDRESSING MODES WHICH INVOLVE ADDRESS REGISTERS

Only a restricted set of MC68000 instructions permit an address register destination operand. These instructions usually have an instruction mnemonic that ends in A, i.e. ADDA, MOVEA, SUBA, etc.

MOVE.size <ea>,<ea>

As a destination <ea> can be a data register or a memory location. An address register (An) destination is NOT allowed.

MOVEA.size <ea>,An

The destination for a MOVEA instruction can only be an address register.

The MC68000 handles address calculations in a different manner than simple math calculations:

- if size=word then the source word is sign-extended prior to the calculation
- only word or long word sizes are allowed

Examples of Using Symbols and MOVEA:

```
        ORG       $6000
FT    DS.L       1

      ORG        $7000
      MOVEA.L    FT,A0       ;moves 32-bit long word
                             at $6000 into A0, i.e.
                             (A0)=$1
      MOVEA.L    #FT,A1      ;moves 32-bit long word
                             address $6000 into A0,
                             i.e. (A0)=$00006000
```

The point is that FT=$6000 in the symbol table. If you refer to FT it simply replaces the symbol with the hex number $6000. Hence, the first instruction is really

```
     MOVEA.L     $6000,A0
```

which you know will put the contents of address $6000 into A0. The second instruction is treated as

```
     MOVEA.L     #$6000,A1
```

which you know will put the number $6000 into A1.

MORE ADDRESS REGISTER SPECIFIC INSTRUCTIONS:

MOVEA.L <ea>,An

The MOVEA instruction moves the contents of the source operand, i.e. the contents of <ea>, to the address register An.

LEA <ea>,An

The LEA instruction simply moves the source operand, i.e. <ea>, to the address register. At this point in your knowledge of MC68000 addressing you can think of the LEA instruction treating the <ea> as an immediate constant.

Example:

```
        ORG       $6000
TE      DC        $ABCD

        ORG       $7000
        MOVEA.L   TE,A0       ;will put $ABCD into A0
        MOVEA.L   #TE,A1      ;will put $6000 into A1
        LEA       TE,A0       ;will put $6000 into A0
        LEA       #TE,A0      ;NOT ALLOWED
        LEA       16(TE),A2   ;can compute addresses
                              as will be shown later in
                              course
```

COMMENTS:

1. Use MOVEA to initialize address registers
2. Use LEA to calculate dynamic addresses such as found in arrays.

USING ADDA AND SUBA

These instructions are used to manipulate addresses

ADDA.<size><ea>,An
SUBA.<size><ea>,An

where <size> can only be word or long word. As shown <ea> can be determined by any addressing mode but the destination can only be an address register. If <size> is word, then the source is sign extended to a long word and all calculations and the result are long word.

Examples:

```
ADDA.L      #100,A0
```

The source is immediate. The destination is address register direct as it should be. This instruction adds long word $64 ($\$64=100_{10}$) to the long word contents of A0.

```
SUBA.W      ALPHA,A1
```

Since ALPHA is a label the source is absolute long. The destination is address register direct as it should be. This instruction adds the sign-extended (ALPHA) to the long word contents of A1.

Suppose (ALPHA) = \$8C07 and (A1)=\$0000 A04B. Then, sign-extending (ALPHA) to \$FFFF 8C07 and adding \$FFFF 8C07 to \$0000 A04B gives (A0) = \$0001 1444. If you did not sign-extend you would get the incorrect result (A0) = \$0000 1444.

ADDRESS REGISTER INDIRECT ADDRESSING MODES

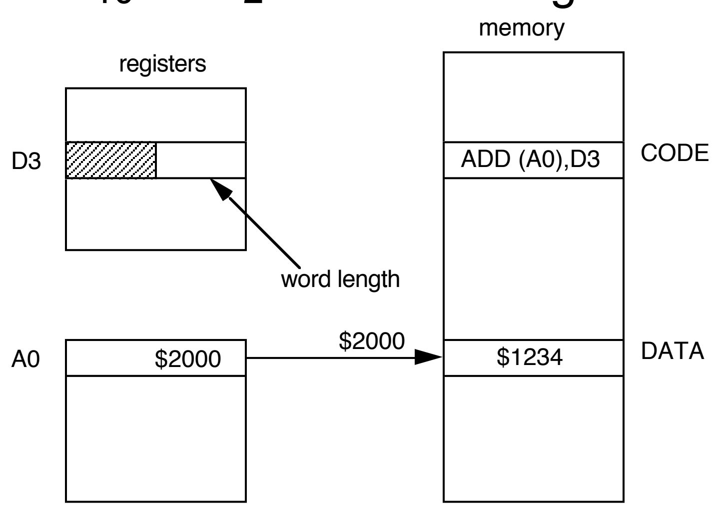
<u>Address register indirect</u>
(mode=010, register=register#)

general form ADD (A0),D3 ← Source is address register indirect

Assembled instruction:

15	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	1	0	0	0	0	1	0	0	0	0
	Opcode	Size			Register			Addressing Mode			Addressing Mode			Register	

where mode=010_2 to indicate address register indirect and register=0_{10}=000_2 to indicate register A0.



This instruction references the contents of the memory location whose address is in A0 adds $1234 to the low 16-bit word contents of D3.

This type of addressing is indicated in the Programmer's Reference Manual by EA=(An)

Address register indirect with **post**increment
(mode=011, register=register#)

general form ADD (A0)+,D3 ← Source is address register indirect with postincrement

Assembled instruction:

15	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	1	0	0	0	0	1	1	0	0	0
Opcode				Size	Register			Addressing Mode			Addressing Mode			Register	

This instruction performs the same ADD but after the ADD is performed will increment the word length contents of A0 by one word (2 bytes). In the previous example this would change the address in A0 from $2000 to $2002

This is indicated in the Programmer's Reference Manual by the notation:

EA=(An)
An←An+N

where N is determined by the instruction size.
N=1 for byte length adds, 2 for word length adds, and 4 for long word length adds.

Address register indirect with **pre**decrement
(mode=100, register=register#)

general form ADD -(A0),D3 ← Source is address register indirect with predecrement

Assembled instruction:

15	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	1	0	0	0	1	0	0	0	0	0
Opcode				Size	Register			Addressing Mode			Addressing Mode			Register	

This instruction performs an ADD but <u>before</u> the ADD is performed the word length contents of A0 are decremented by one word (2 bytes). In the previous example this changes the address in A0 from $2000 to $1FFE and does a word length add of the contents of $1FFE to D3.

The manipulation of the source is indicated in the Programmer's Reference Manual by the notation:

An←An-N
EA=(An)

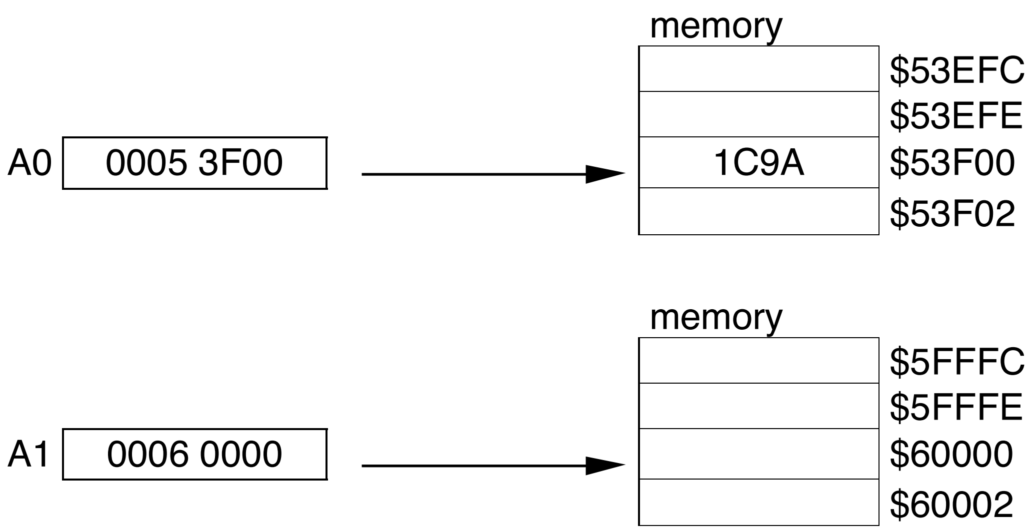
where N is determined by the instruction size.
N=1 for byte length adds, 2 for word length adds, and 4 for long word length adds. Note that the subtraction

is shown before the effective address to indicate that that contents of the address register are decremented and then used to determine the source address.

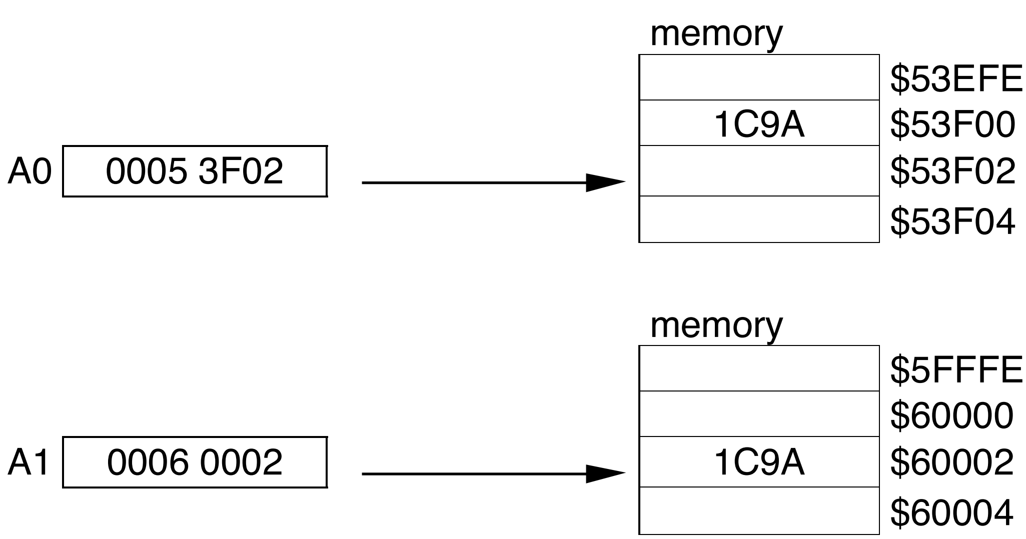
interesting forms of combining indirect addressing:

MOVE.W (A0)+,(A1)+

Before:



After:



These modes are good for moving blocks of data from one area of memory to another.

More interesting instructions:

MOVE.W (A0)+,(A1)+

Copies one data word from (A0) to (A1)

1. reads word contents of (A0)
2. increments A0 by 2 bytes
3. moves word to (A1)
4. increments A1 by two bytes

MOVE.W -(A0),-(A1)

Copies one data word from (A0) to (A1)

1. decrements A0 by 2 bytes
2. reads word at new (A0)
3. decrements A1 by 2 bytes
4. writes word at new (A1)

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

Copies one data word from (A0-2) to (A0-4)

1. decrements A0 by 2 bytes
2. reads word at new (A0)
3. decrements A0 by 2 bytes
4. writes word to new (A0), i.e. original A0 - 4 bytes

MOVE.L (A2)+,(A2)

Copies one long word from (A2) to (A2+4)

1. read long word at (A2)
2. increments A2 by 4 bytes
3. writes long word to new (A2)

MOVE (A7)+,(A7)

Copies the word at (A7) to (A7+2)

The above MOVE instructions only affect memory; no data registers are affected.

ADD.W (A0)+,D0

Adds word at (A0) to D0

1. reads word at (A0)
2. increments A0 by 2 bytes
3. adds word to D0

MOVE.W -(A2),D0

Adds word at (A2-2) to D0

1. decrements A2 by 2 bytes
2. read word at new (A2)
3. writes word to D0

MOVE.W D1,-(A1)

Moves a word from D1 to the new (A1) after A1 is decremented

MOVE.W (A1)+,D2

Moves (A1) to D2; A1 is incremented by 2 bytes

A simple rule to remember for computing addresses is to evaluate the expression from left to right.

<u>Address register indirect with index and 8-bit displacement*</u>
(mode=110, register=register#)

*also called offset by Motorola

Examples

ADD	4(A0,D6),D3	← uses A0 as the base address
ADD	LABEL(A0,D6),D3	← uses the symbol LABEL as the base address

Assembled format of instruction:

15	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	1	0	0	0	1	1	0	0	0	0
(1)	Rn			(2)	0	0	0	8-bit displacement							

Notes:
Rn index register number, 0≤Rn≤7
(1) type of index register, 0=data register, 1=address register
(2) size of index register for address computation, 0=word length, 1=long word length
8-bit displacement is 2's complement number

The source address is computed as the sum of the base address (the contents of A0, in these examples), the displacement, and the offset. The addition is performed using all 32-bit numbers and the result is a 32-bit address. The index (either an address or data register) is either a 32-bit number or a 16-bit number sign extended to 32 bit. The displacement is an 8-bit number in 2's complement notation sign extended to 32 bits for the address calculation.

```
ADD    4(A0,D6.W),D3     case 1
ADD    4(A0,D6.L),D3     case 2
```

case 1:

A0 (the base)

sign extension | index (.W)

sign extension | displacement

---

the calculated address

In this case the word-length index register is sign extended to a long word, and the byte-length displacement is extended to a long word. The actual address calculation is performed using all long word numbers.

case 2

A0 (the base)

index (.L)

sign extension | displacement

the calculated address

In this case the index register is long word needing no sign extension, and the byte-length displacement is extended to a long word. The actual address calculation is again performed using all long word numbers.

Example:
Using address register indirect addressing to access a table of numbers.

A MC68000-based system monitors four pressure valves in a chemical processing plant. Each valve's pressure is recorded every half-hour. These readings are sequentially stored in memory as shown below.

```
MOVE.W          VALVE(A0,D0.W),D1
```

retrieves any selected valve reading into D1 where:

A0	contains the beginning address of the table, in this case $53F00
D0.W	contains the number of the reading, in this case n=8
VALVE	is the reference to a particular valve. In this case VALVE=4.