ADDRESSING MODES

An addressing mode tells the computer where to get/place a number.

Basic form of a MC68000 instruction:
Instruction Source, Destination

The source and destination can use DIFFERENT addressing modes.

Addressing modes on the MC68000 are usually specified in the first 12 bits of the 68000 instruction word:

```
 1 5 111119 8 7 6 5 4 3 2 1 0
 Opcode  Size Destination Addressing Mode Source Addressing Mode Register
```

A few specialized instructions make assumptions about the addressing modes and use different formats. For example, MOVEQ assumes that the source is an 8-bit immediate constant and the destination is a data register.

MOVEQ
|   |   |   |   |   | 1 | 1 | 1 | 1 | 1 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | Destination | Register | Dn | 0 |   |   |   |   | 8-bit constant | (-128 to +127) |   |   |   |   |   |   |
For the following examples we will consider the different source addressing modes for a word length ADD instruction which will put the resulting word into data register D3.

The general form of the ADD instruction:

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

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Register</th>
<th>Op-mode</th>
<th>Effective Address Mode</th>
<th>Register</th>
</tr>
</thead>
</table>

The general form of the ADD instruction which adds the contents of D3 to something and puts the results into D3 is:

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

where

- bits 15-12 indicate the op code 1101 for an ADD
- bits 11-9 indicate that data register D3 (i.e. 3) is the destination for the result of the ADD ($3_{10}=011_2$)
- bits 8-6 indicate the op-mode of the ADD. In this case the calculation will be a word length ADD of the form $<Dn>+(<ea>)\rightarrow<Dn>$. This is indicated by 001; see the Programmer’s Reference Manual for information about the other modes. [NOTE: This translates into add the contents of D3 to the contents of the effective address $<ea>$ and put the result into D3.]
- bits 5-3 indicate the addressing mode of the source
• bits 2-0 indicate the register (if applicable) of the source

We will examine bits 5-0 in detail in the following examples.
Source is data register direct  
(mode=000, register=register#)

general form  ADD      D1,D3  ← Both source and  
destination are  
data register  
direct

Assembled instruction:

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

where mode=000₂ to indicate a data register and  
register=1₁₀=001₂ to indicate register D1.

Address register direct  
(mode=001, register=register#)

general form  ADD      A1,D3  ← Source is address  
register direct;  
destination is data  
register direct

Assembled instruction:

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

where mode=001₂ to indicate an address register and  
register=1₁₀=001₂ to indicate register A1.
Immediate (mode=000, register=100)
Although this looks like an ordinary ADD instruction

general form  ADD  #7,D3  ← Source is immediate; destination is data register direct

this instruction is always re-coded by the assembler into the more specific ADDI (ADD immediate) instruction format

general form  ADDI  #7,D3  ← Source is immediate; destination is data register direct

which has the instruction format:

<table>
<thead>
<tr>
<th>15</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>Size</th>
<th>Effective Address Mode</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Word Data (16 bits)</td>
<td>Byte Data (8 bits)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Long Data (32 bits, including previous word)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where the immediate constant is stored in one or two extension words according to the above conventions.
In this particular case the assembled instruction will become:

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

where the constant is stored in one extension word. Note that it is the instruction (default is .W) which determines the mode, NOT the size of the constant.
NOTE: As shown the instruction ADDI #7,D3 takes two words of computer memory. This instruction could be shortened to one word using the ADDQ (ADD quick) instruction. A quick instruction uses a special instruction word format which can include constants represented as 3-bit binary numbers (the constants are limited to the range 1 to 8 decimal)

general format:

<table>
<thead>
<tr>
<th>15</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>Data</td>
<td>0</td>
<td>Size</td>
<td>Effective Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td>Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assembled instruction:

<table>
<thead>
<tr>
<th>15</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

where the size = 01₂ to indicate a word length operation and the effective address is that of the source. In this case it is mode=000₂ to indicate a data register and register=011₂ to indicate D3.

Note that if the immediate constant was 1234567810 that the ADDQ instruction could NOT be used. The ADDI with two (2) extension words would be used because the binary constant is so large.

Different methods of specifying immediate constants:

```
ADD.W #$452,D3 ;specifies a hexadecimal constant
```
ADD.L   #I,D3  ;specifies the address of I as a constant
ADDI.W  #%1011,D3 ;specifies a binary constant
IMMEDIATE ADDRESSING

#xxx indicates immediate addressing in the Programmer’s Reference Manual

Immediate mode addressing can only be used for source addressing; it is NOT allowed for destination addressing.

Examples of immediate mode addressing:

MOVE.W #$452,D0 ;moves the number $452 to D0

MOVE.L #I,D0 ;moves the value of I (the address in the symbol table) into D0 as a number

ADDI.W #%1011,D0 ;add the binary constant 1011 (13\text{\textsubscript{10}}) to the contents of D0

MOVE.L I,D0 ;move the contents of the memory location I into D0. NOT IMMEDIATE MODE ADDRESSING.

;Yes, you can do calculations with labels
ABSOLUTE MODES OF ADDRESSING

absolute


xxxxxx indicates absolute long addressing in the Programmer’s Reference Manual

Whether an absolute addressing mode is long or short is usually assigned by the assembler.

eamples of absolute long addressing:

MOVE.W I,D0 ;moves the contents of address (longword) I to D0

ADD.W $500,D0 ;add the contents of $0000 0500 to D0

MOVE.W D0,I-4 ;move the contents of D0 to (long word) address I-4

;Yes, you can do calculations with labels

eamples of absolute short addressing:
(usually only the assembler does this)

MOVE.W $1000.W,D0 ;address is sign extended to 24 bits and stored in a single extension word
ADD.L I.W,D0 ;takes only the lower 16 bits of I to form the extension word
Absolute long addressing is generated by default. Absolute short addressing is usually generated by the assembler rather than directly by the programmer. The assembler does this to generate shorter code but it restricts the memory range you can access.

As absolute short uses a 16-bit extension word it can only be used to access memory at the bottom of memory or at the top of memory as shown below. Absolute short cannot be used to access the memory in between.
ADD.L I,D3 ;not immediate, specifies moving the contents of I (I is a label)

I DC.L 75 ;I specifies a memory address, the DC.L instruction instructs the assembler to reserve 75 longwords in memory beginning at this address

absolute long
(mode=111, register=001)

The primary difference between Absolute Short and Absolute Long is that the address is bigger than 16-bits and must be represented as a full 32-bit address.

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

The assembled form of an absolute long ADD instruction is:

```
1 1 1 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
x x x x x x x x x x x x x x x x
```

where xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx is a 32-bit binary address indicating 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.