68k Addressing Mode Examples

John Winans

1 Overview

The addressing mode of an instruction can specify the value of an operand (immediate), a register that contains the operand (register direct), or how the effective address (EA) of an operand in memory is derived. [1, Section 2.4]

Note that the 68k family of CPUs are big-endian and have 8 data registers and 8 address registers. Register A7 is used as the program stack pointer.

2 Examples

For each example that follows, assume the state of the machine is:

<table>
<thead>
<tr>
<th>Data Registers</th>
<th>Address Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0 = 0x12345678</td>
<td>A0 = 0x00001000</td>
</tr>
<tr>
<td>D1 = 0x00000004</td>
<td>A1 = 0x000000a0</td>
</tr>
<tr>
<td>D2 = 0x00000001</td>
<td>A2 = 0x00000050</td>
</tr>
<tr>
<td>D3 = 0xff00ff00</td>
<td>A3 = 0x33123456</td>
</tr>
<tr>
<td>D4 = 0x00ff00ff</td>
<td>A4 = 0x00000000</td>
</tr>
<tr>
<td>D5 = 0xd5333333</td>
<td>A5 = 0x00000000</td>
</tr>
<tr>
<td>D6 = 0x88888888</td>
<td>A6 = 0x0000008c</td>
</tr>
<tr>
<td>D7 = 0x00000000</td>
<td>A7 = 0x000000a0</td>
</tr>
</tbody>
</table>

PC=0x00000100

00000000 8d 3a a8 cb 7d 31 5e a1 93 a5 61 45 00 00 00 80 |.:..}1^...aE....|
00000010 a5 98 ad c8 b9 a0 c3 c8 17 2b 9e c8 9b b2 70 ff |............+.....p.|
00000020 aa 2d 13 31 c1 34 d7 fd 18 13 cc 01 53 db fb 7b |.-.1.4.......S..{|
00000030 1c db a6 7b 19 f6 aa fe 59 76 0c 87 75 04 48 57 |...{'....Yv..u.HW|
00000040 1b e0 92 b5 96 0d 0f d8 fd c7 b6 82 05 56 89 e9 |..............V..|
00000050 33 21 83 7a 50 e2 ee 3e db f6 e0 0f de 63 fc c4 |3!..zP..>.....c...|
00000060 1d 48 52 3f 28 36 29 aa 5d 66 d9 41 7c 33 62 b9 |..HR?(6)..]f.A|3b.|
00000070 fd bc d6 fa a2 32 b8 d8 a0 13 1c ba 1b ef 93 96 |....2..........|
00000080 75 68 19 f3 2d 13 ba 27 dc 16 51 a9 65 ff fd 86 |uh..~’...Q.e...|
00000090 d7 04 d0 72 15 ab 8b 89 e3 4d 86 f2 00 00 00 10 |...r.....M......|
2.1 Address Register Direct

The operand is in an address register.

For example if an instruction uses this mode and specifies register A3 then the operand value will be 0x33123456

2.2 Data Register Direct

The operand is in a data register.

For example if an instruction uses this mode and specifies register D5 then the operand value will be 0xd5333333

2.3 Address Register Indirect

The operand is in memory at the address provided by the value in an address register.

For example, given the machine state shown above, if an instruction uses this mode and specifies register A2 then the operand value will be 0x3321837a (fetched from memory address 0x00000050.)

2.4 Address Register Indirect with Postincrement

This mode can be used as part of a POP operation from a full descending stack.

The operand is in memory at the address provided by the value in an address register. After the operand is fetched from memory, the address register is incremented to point to the “next” item in the stack “below” the item just popped.

For example, given the machine state shown above, if an instruction uses this mode to read a 32-bit value from memory and specifies register A2 then the operand value will be 0x3321837a (fetched from memory address 0x00000050) and A2 will be changed to 0x00000054.

2.5 Address Register Indirect with Predecrement

This mode can be used as part of a PUSH operation into a full descending stack.

The operand is in memory at the address provided by the value in an address register after said address register is decremented to point to a place to store the “new” item in the stack “above” the current/previous top.

For example, given the machine state shown above, if an instruction uses this mode to write the 32-bit value 0x11223344 into memory and specifies register A2 then the memory at address 0x0000005c will be overwritten with the value 0x11223344 and A2 will be changed to 0x0000005c.
2.6 Address Register Indirect with Displacement

The operand is in memory at the address calculated as the sum of an address register and a displacement value.

For example, given the machine state shown above, if an instruction uses this mode to read a 32-bit value from memory and specifies register A2 and a displacement value of 0x0000000c then the operand value 0xde63fcc4 will be read from address 0x0000005c.

2.7 Address Register Indirect with Index and (8-32 bit) Displacement

The operand is in memory at the address calculated as the sum of an address register, an index register (for this we can use a data or address register) and a displacement value. Note that the index register will be multiplied by either 1, 2 or 4 before it is included in the sum. (IBM Mainframe assembler expresses these as D(X,B) and does not scale the X register.)

For example, given the machine state shown above, if an instruction uses this mode to read a 32-bit value from memory and specifies register A2 for the base, D1 for the index, 4 for the scale, and a displacement of 0x10 then the memory address will be calculated as:

0x00000000 + (0x00000004 * 4) + 0x10 = 0x00000070

... and the operand value fetched will be 0xfdbcd6fa.

2.8 Memory Indirect Postindexed

The operand and its address are both stored in memory!

The operand address is specified using a similar method as discussed for Address Register Indirect with Index and Displacement above. However the final memory address of the operand is calculated using by a value that is stored in memory... at the address calculated by adding the value of an address register to a displacement value.

For example, given the machine state shown above, if an instruction uses this mode to read a 32-bit value from memory and specifies register A6 for the base, 0x10 for the base displacement, D1 for the index, 4 for the scale, and an outer displacement of 0x08 then the memory address will be calculated as:

indirect address = 0x00000008c + 0x10 = 0x0000009c
indirect value = 0x00000010
operand address = 0x00000010 + (0x00000004 * 4) + 0x08 = 0x00000028

... and the operand value fetched will be 0x1813cc01.

2.9 Memory Indirect Preindexed

This is the same sort of thing as Memory Indirect Postindexed but the index register is applied during a different part of the calculation.

The address of the value used for the indirect is calculated using the same logic as described in
Address Register Indirect with Index and Displacement. The indirect value is then fetched and added to the outer displacement and used as the final address at which the operand is fetched.

For example, given the machine state shown above, if an instruction uses this mode to read a 32-bit value from memory and specifies register A6 for the base, 0xffffffff7c for the base displacement, D2 for the index, 4 for the scale, and an outer displacement of 0x08 then the memory address will be calculated as:

\[
\text{indirect address} = 0x0000008c + 0xffffffff7c + (1 \times 4) = 0x00000008 + 4 = 0x0000000c \\
\text{indirect value} = 0x00000080 \\
\text{operand address} = 0x00000080 + 0x08 = 0x00000088 \\
\ldots \text{and the operand value fetched will be } 0xdc1651a9.
\]

2.10 Program Counter Indirect with Displacement

This is the same operation as Address Register Indirect with Displacement but uses the current PC register value for the base address (as opposed to a value in an address register.)

One oddity is that the PC register value used in the calculation of the operand address is the value that it has at the time the address calculation is taking place in the CPU. For purposes of this example, let us assume that it is pointing at the first (of a potential plurality) extension word. That is, the address that is 2 past the address of the instruction being executed. (Note: This is not the case for RISC-V instructions. However, it is very common in PC relative addressing modes on most processors.)

For example, given the machine state shown above, if an instruction located at the address in the PC register uses this mode to read a 32-bit value from memory and specifies displacement value of 0xffffffff82 then the operand address will be calculated as:

\[
\text{operand address} = 0x00000100 + 2 + 0xffffffff82 = 0x00000084 \\
\ldots \text{and the operand value fetched will be } 0x2d13ba27.
\]

2.11 Program Counter Indirect with Index and (8-32 bit) Displacement

This is the same operation as Program Counter Indirect with Displacement but includes a scaled index register in the calculation.

For example, given the machine state shown above, if an instruction located at the address in the PC register uses this mode to read a 32-bit value from memory and specifies displacement value of 0xffffffff82, D1 for the index register, and 4 for the scale then the operand address will be calculated as:

\[
\text{operand address} = 0x00000100 + 2 + 0xffffffff82 + (0x00000001 \times 4) = 0x00000088 \\
\ldots \text{and the operand value fetched will be } 0xdc1651a9.
\]
2.12 Program Counter Memory Indirect Postindexed

This is the same operation as Memory Indirect Postindexed but uses the current PC register value for the base address (as opposed to a value in an address register.)

2.13 Program Counter Memory Indirect Preindexed

This is the same operation as Memory Indirect Preindexed but uses the current PC register value for the base address (as opposed to a value in an address register.)

2.14 Absolute Addressing

In this mode the operand is in memory at an address stored in an immediate value.

For example, given the machine state shown above, if an instruction uses this mode to read a 32-bit value from memory and specifies an immediate value of 0x00000088 then the operand fetched will be 0xdc1651a9.

2.15 Immediate

In this mode the operand is in an immediate value.

For example, given the machine state shown above, if an instruction uses this mode and and specifies an immediate value of 0x00000088 then the operand value is 0x00000088.

References