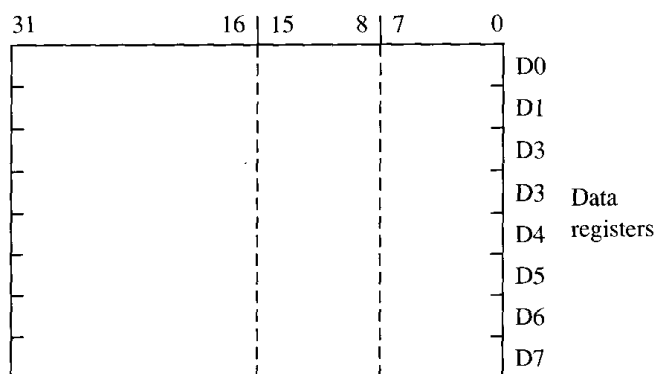


MC68000 PROGRAMMING MODEL

The programming model for the MC68000 microprocessor is shown in Figure. It consists of

- 19 registers
- 16 M-byte (2^{24} bytes or 2^{23} 16-bit words) combined memory
- I/O space.

The bank of eight 32-bits data registers, D0 through D7, identifies the MC68000 as a general register machine.

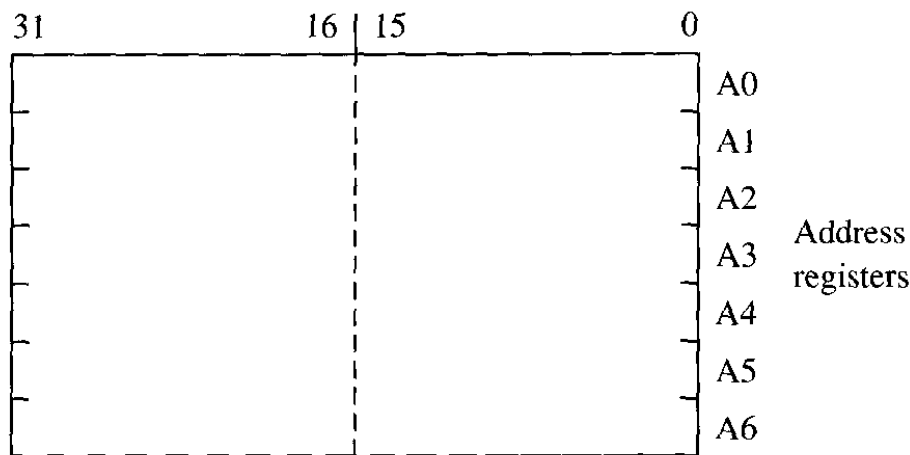


The data movement and arithmetic/logic instructions manipulate data in these registers or in memory locations. Most of those instructions can be specified to operate on 8-, 16-, or 32-bit pieces of data in the data registers. Eight-bit operations affect only bits 0 through 7, 16-bit operations affect bits 0 through 15, and 32-bit operations affect the entire register.

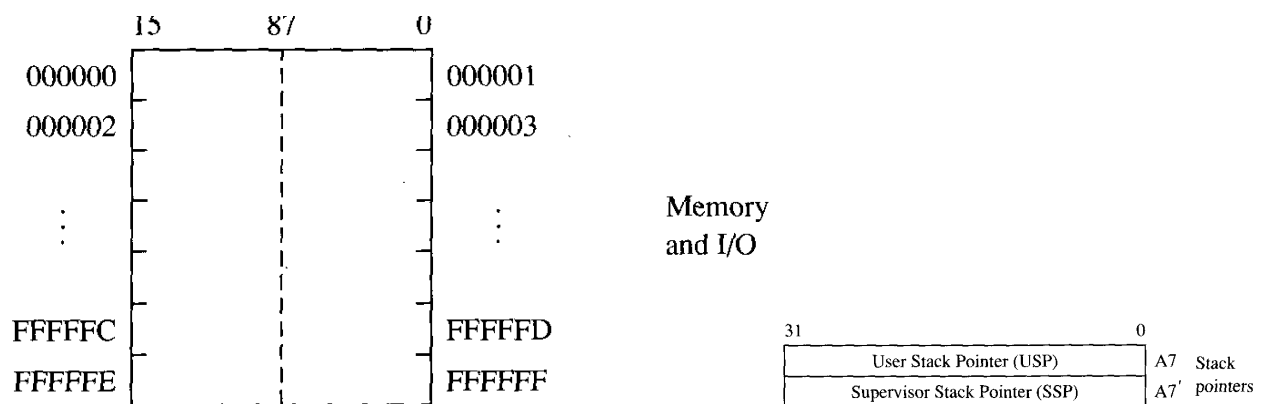


Although the address registers and the program counter are 32 bits wide, the processor chip has a 24-bit address bus, which uses only the lowest ordered bits of these registers. Thus, the processor is capable of addressing 2^{24} individual bytes (16MB).

Registers A0 through A7 and A7' are used to hold addresses of locations in memory which are of particular interest in a program.

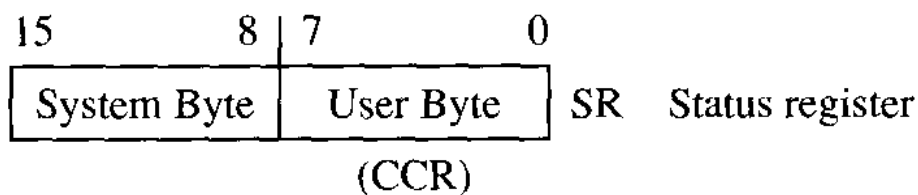


A byte may be accessed at any one of 16M locations, but a word (16-bits) may be accessed only at one of the 8M even-addressed locations.



The internal structure and the instruction set of the MC68000 support 32-bit operations; however, its 16-bit data bus and its byte-addressing scheme limit memory accesses to 8 or 16 bits of data per cycle.

The status register (SR) is divided into two parts, the *user byte* and the *system byte*. The system byte will be discussed later. The user byte (also referred to as the CCR) is directly affected by the execution of certain instructions. It includes the usual carry bit (C), overflow bit (V), zero bit (Z), and sign or negative bit (N). In addition, it includes the extend bit (X), a modified version of the carry bit which is used exclusively for multi-precision arithmetic operations



A FIRST EXAMPLE PROGRAM FOR THE MC68000

The column to the right of the memory content column lists the mnemonic form of each instruction after the final word of that instruction. The last two columns show the contents of the PC and either register D0 or D7 after the word is fetched or the instruction is executed. The dashes in the contents of the data registers indicate that the values present are immaterial because they either will not be affected by any instruction or they will be overwritten by the current instruction.

<i>Address</i>	<i>Content</i>	<i>Instruction</i>	After Fetch (Execute)	
			<i>PC</i>	<i>D0</i>
001000	3038		001002	-----
001002	1014	MOVE 1014H,D0	001004	----1234
001004	D038		001006	----1234
001006	1016	ADD 1016H,D0	001008	----68AC
001008	3E00		00100A	----68AC
00100A	1018	MOVE D0,1018H	00100C	----68AC
00100C	2E3C		00100E	D7
00100E	0000		001010	-----
001010	00E4	MOVE L #228,D7	001012	000000E4
001012	4E4E	TRAP #14	001014	000000E4
001014	1234	Data		
001016	5678	Data		
001018	0000	Place for result		

A simple MC68000 program segment.

The PC starts at 001000, but after the first word is fetched it is incremented to 001002.

The processor starts in the fetch cycle and will therefore fetch the content of the word location in memory to which the program counter is pointing. It does this by placing the address 001000 on the address bus and executing a read cycle. It then automatically increments the program counter by two. The word fetched is 3038, and, since it is supposed to be the first word of an instruction, the processor decodes it as such.

Having completed fetching the instruction, the processor proceeds to execute it.

The instruction, 3038-1014 (MOVE 1014H,D0), requires that the processor fetch the word from the memory location whose address is 1014, sign extended to 001014. The processor is then to load that word into register D0. The processor does this by executing another read cycle, this time with the address 001014, and latching the value read (1234) into D0. Notice that this is a word (16 bits) transfer, so it does not affect the most significant half of D0.

The second instruction, D038-1016 (ADD 1016H, D0), calls for the processor to add to the content of register D0 the word read out of location 1016 (sign extended to the 24-bit address 001016). This requires another read cycle to obtain the word from memory location 001016 (5678 hex). The processor adds this value to the content of D0 to generate the sum of 68AC.

The third instruction, fetched from locations 001008 and 00100A, is 3E00-0018 (MOVE D0,1018H). This instruction tells the processor to copy the content of register D0 into memory location 001018. It does so by executing a memory write cycle with the address 001018 and the data 68AC.

The fourth instruction (MOVE.L #228,D7) is a three-word instruction (2E3C-0000-OOE4 in hex) used to prepare the system to terminate the program segment gracefully. It requires the processor to load the 32-bit pattern for the decimal number 228 (hex 000000E4) into register D7. When the final instruction (4E4E or TRAP #14) is executed, it acts as a call to the monitor or the operating system and, in conjunction with the value stored in D7, serves to end the program. All programs which use the ASSYM000 simulator software must end with these two instructions.

THE MC68000 ADDRESSING MODES

This section describes the addressing modes used in the Motorola MC68000 microprocessor, their formats within program memory, and how they are specified in the instruction mnemonics.

Motorola's literature lists 14 different addressing modes used with the MC68000. One of these, the implied mode, is not discussed here since it requires no specification on the part of the programmer.

The remaining 13 are grouped into seven categories in this chapter:

- absolute,

- immediate,

- register direct,

- address register indirect,

- address register indirect with auto increment/decrement,

- address register indirect with offset/index, and

- PC relative.

Instructions occupy one to five consecutive (16-bit) words in memory. The addressing mode is encoded in the first word, the op-word, which also identifies the total number of words in the instruction and the operation it is to implement. The remaining words, called extension words, further specify the operands when necessary.

Although the MC68000 is described as a 16-bit processor and has a 16-bit data bus, it can access memory on the byte level. That is, every byte in the memory space has its own 24-bit address. Instructions may refer to a single byte, a double-byte word, or a quadruple-byte long word.

Memory:

Even Addresses			Odd addresses
n	LW0, W0, B0	---, --, B1	$n + 1$
$n + 2$	---, W1, B2	---, --, B3	$n + 3$
$n + 4$	LW1, W2, B4	---, --, B5	$n + 5$
$n + 6$	---, W3, B6	---, --, B7	$n + 7$

FIGURE MC68000 data formats.

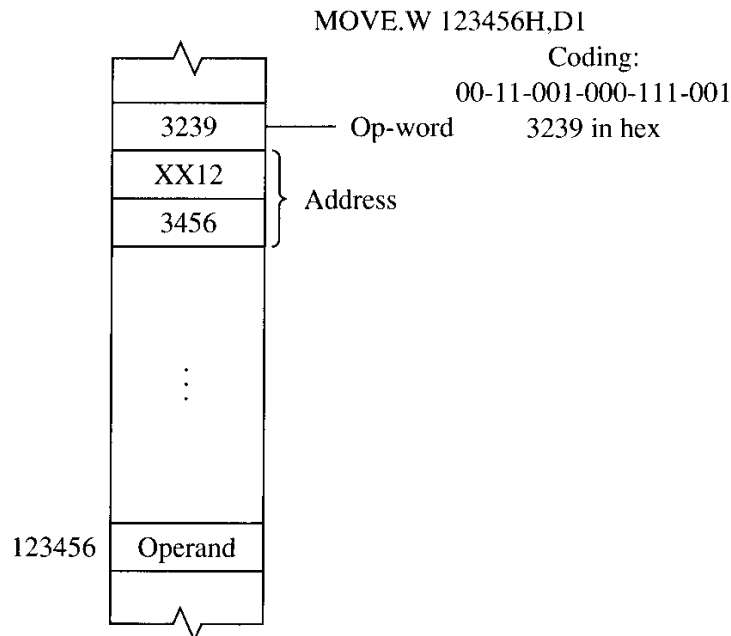
The figure shows eight consecutive bytes in memory which may be accessed as eight bytes, four words, or two long words. The most significant byte is B0 and it comes first in memory. The most significant word is W0 and it comes first and includes bytes B0 and B1. The most significant long word is LW0 and it comes first and includes bytes B0 through B3 as well as words W0 and W1. Bytes may be accessed at any location, but words and long words may be accessed only at even addresses as shown.

With most instructions the programmer must select the desired data size by including an extension after the mnemonic (.B .W or .L). If such an extension is not included, the default size is word.

1 ABSOLUTE ADDRESSING IN THE MC68000

The MC68000 includes two forms of absolute addressing, long and short. The long form requires two extension words (32 bits) after the op-word to specify the 24-bit address (the most significant 8 bits have no effect). The short form specifies a 16-bit address in a single extension word. During execution it is sign-extended to 32 bits (24, effectively)

Figure shows an example of the absolute long mode. The instruction is to load register D1 with the (word) content of location 123456 hex. The mnemonic form is MOVE.W 123456H,D1 including the optional .W extension. The source and destination identifiers are listed in the *order from, to* (from location 123456 to register D1).



Absolute long addressing in the MC68000.

The op-word is found from the tables of Appendix C to be

00 - size - destination - source (effective address)

The size code for word is 11, the destination code for register D1 is 001,000 (register number, mode), and the source or effective address code for absolute long is 111,001 (mode, register in the table). Combining these yields 00 11 001 000 111 001, which is 3239 in hex. This op-word is followed by the two-word extension, which provides the absolute address. In Figure the symbol X represents nibbles which are of no consequence in the program but which would normally be made equal to 0.

In mnemonic form the same instruction with a 32-bit operand would be specified by MOVE.L 123456H.D1. When this instruction is executed the processor loads the word from location 123456 into the upper half of D1 and the word from location 123458 into the lower half of D1.

2 IMMEDIATE ADDRESSING IN THE MC68000

Immediate addressing requires that the instruction include the operand as an integral part of itself (in extension words). Thus, this addressing mode refers to locations immediately following the op-word.

An example of immediate addressing is shown in Figure, where the instruction is

to move the 16-bit word 789A into the low half of D1. Immediate addressing in Motorola processors is indicated in the mnemonic form by the use of the number symbol #. The example MOVE #789AH,D1 implies a word length operand by the lack of an extension and should be read as "move the word-sized number 789A hex into register D1". The op-word is 323C, which is followed by the single extension word, which contains the operand itself.

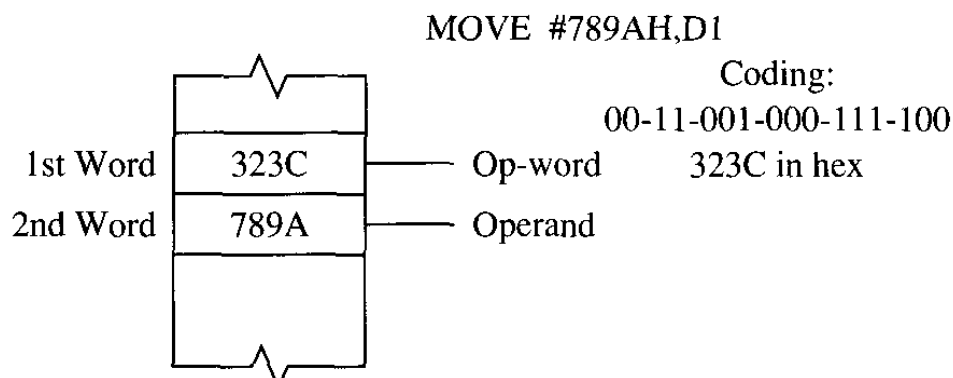


FIGURE Immediate addressing in the MC68000.

3 REGISTER DIRECT ADDRESSING IN THE MC68000

In the register direct modes the operand is in a processor register..

<i>Instruction</i>	<i>Binary coding</i>	<i>Hex code</i>
a. MOVE D5,D1	00-11-001-000-000-101	3205
b. MOVE A3,D1	00-11-001-000-001-011	320B
c. MOVEA D1,A3	00-11-011-001-000-001	3641

When the *destination* for a MOVE instruction is an *address register* then a different mnemonic, MOVEA, is used.

4 ADDRESS REGISTER INDIRECT ADDRESSING IN MC68000

The address register indirect mode calls upon the processor to use an address register as a pointer to a target memory location. This mode must specify one of the address registers (not a data register) as a pointer. Register indirect addressing in the MC68000 includes many options which will be discussed later. These options are listed in Motorola's specifications as separate addressing modes.

Three examples of instructions using the register indirect mode of addressing to specify a source or destination are shown in Figure. Each is encoded with a single op-word and no extensions.

The first instruction shown in Figure 5.25 should be read as "move the (word) content of the memory location pointed to by register A3 into data register D1".

<i>Instruction</i>	<i>Binary coding</i>	<i>Hex</i>
MOVE (A3),D1	00-11-001-000-010-011	3213
MOVE D1,(A3)	00-11-011-010-000-001	3681
MOVE (A3),(A6)	00-11-110-010-010-011	3C93

FIGURE Address register indirect addressing in the MC68000.

The second example, "move the (word) from D1 into the memory location pointed to by A3," is an example of a store operation, where data is moved into memory from the processor.

The third example is a rarity among microprocessors, a memory-to-memory move which bypasses the programming model registers. Its operation is to move the content of a location pointed to by A3 into the location pointed to by A6.

5 ADDRESS REGISTER INDIRECT WITH AUTO INCREMENT/DECREMENT IN THE MC68000

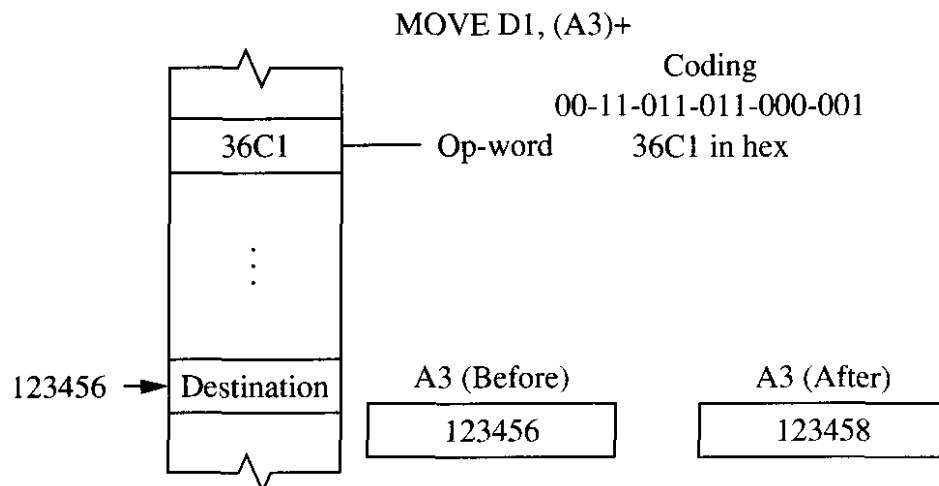
Arrays of data are normally stored in sequential locations in memory. When operating on arrays, the processor often traverses a program loop during which it must change the address in an address register in order to "point to" subsequent array elements. In this way the array may be accessed one element at a time during each pass through the loop. It may be necessary to increment or decrement the address, depending upon the direction in which the array is scanned. The address may need to be changed by 1, 2, or 4, depending upon whether the array consists of bytes, words, or long words.

The auto increment (decrement) mode first uses the register as a pointer in the instruction and then increments (decrements) its value. To emphasize the sequence of activities the modes are referred to as *address register indirect with postincrement* and *address register indirect with predecrement*, often abbreviated *post-inc* and *pre-dec*. In the mnemonic form the modes are indicated by the use of a plus or minus sign.

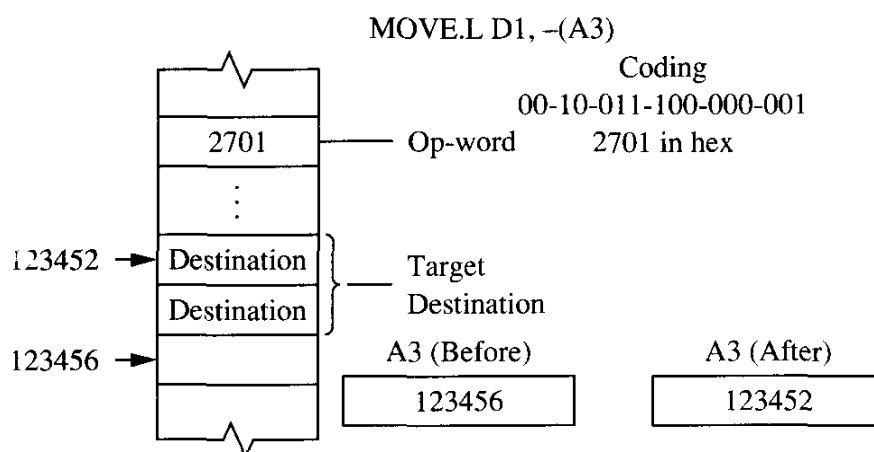
The amount of increment or decrement is automatically selected by the processor during execution to reflect the data size specified in the instruction: 1 for byte, 2 for word, and 4 for long word. The one exception to this is when the pointer is register A7. In that case a byte operand will result in an increment or decrement of 2 in order

to keep the register pointing to a word boundary. This is necessary because of the special use of A7 as a stack pointer, as will be described later.

The first example is the instruction `MOVE D1,(A3)+`, move the word in D1 to memory, address register A3 indirect with postincrement. When this single word instruction is executed the processor moves a word from D1 into the location pointed to by A3 and then increments the content of A3 by 2.



The example in Figure b is the instruction `MOVE.L D1,-(A3)` ("move the long word in D1 to memory, address register A3 indirect with predecrement"). When it is executed the processor first decrements the content of register A3 by 4. This opens up four new bytes in memory where the long word in D1 is subsequently stored in accordance with the instruction.



As a consequence of using the increment/decrement modes not only does the instruction accomplish its primary task of manipulating the targeted operands, it also modifies the content of the selected address register. Thus, these modes combine two basically different operations into a single instruction.

6 ADDRESS REGISTER INDIRECT WITH OFFSET/INDEX IN THE MC68000

Based addressing, builds upon the address register indirect mode by allowing the programmer to specify a fixed offset value to be temporarily added to the content of the register in order to generate the target address. It is available in the MC68000 where it goes by the name *address register indirect with displacement*. The displacement or offset value is a 16-bit signed number which occupies a single extension word within the instruction. When the instruction is executed the processor temporarily adds a sign-extended 32-bit version of the displacement, together with the contents of the specified address register, to generate the target address. The content of the address register is not changed by the instruction.

An example of this mode is shown in Figure. The displacement value of 4000 hex is specified in the mnemonic form by writing it just ahead of the lead parenthesis indicating the register indirect mode: `MOVE D1,4000H(A3)`. The displacement must be an even number when the operand size is word or long word so that the target address will also be an even number.

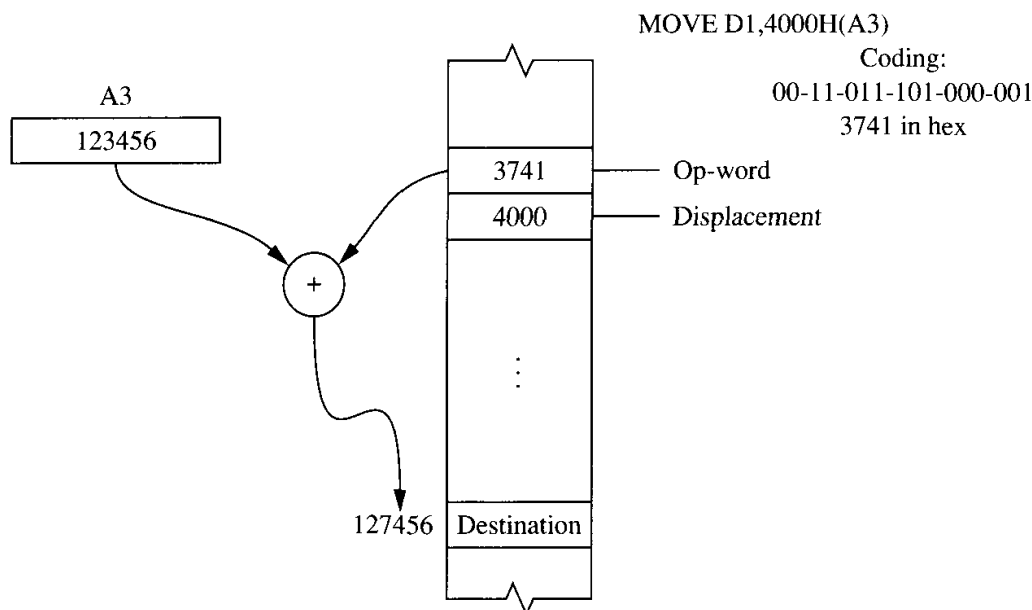


FIGURE 5.27 MC68000 address register indirect addressing with displacement.

A combined indexed and based addressing mode is available in the MC68000 under the name *address register indirect with index*. This mode allows the programmer to specify both a constant offset and a processor register to be used as an index register. When this mode is used, the processor adds three numbers together to generate the target address. or data register which the instruction has specified as the index register.

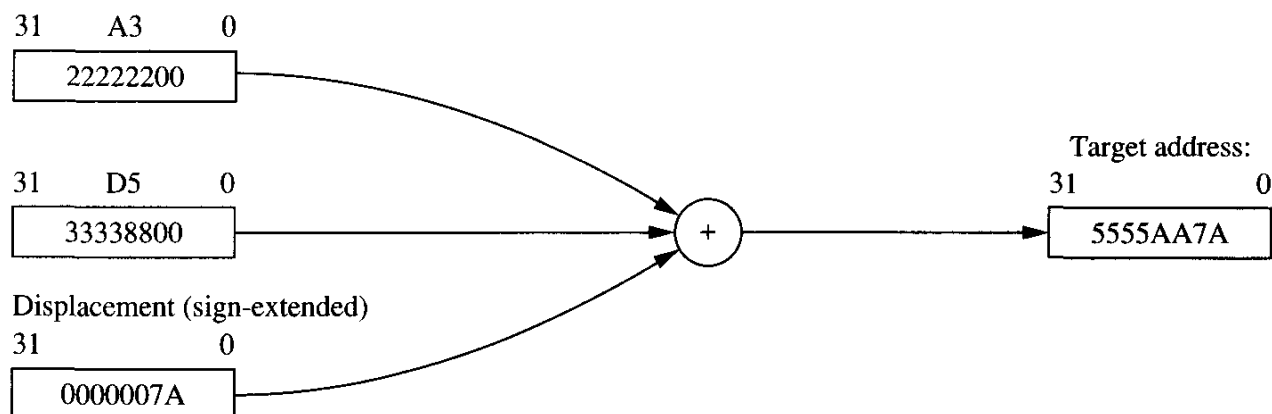
The instruction `MOVE D1,7AH(A3,D5.L)` is shown bellow.

`MOVED1,7AH(A3,D5.L)` Coding: ~~00-11-011-110-000-001~~=3781

~~MOVE.W A3 Mode Mode D1~~

The extension word includes all the necessary index and displacement information, including the eight-bit offset value of 7A. An instruction which uses this mode for both the source and the destination would require two extension words, one for the source and one for the destination.

The calculation of the target address during the execution of the example instruction is shown in Figure. Registers A3 and D5 are assumed to contain the values 22222200 and 33338800, respectively. The processor generates the target address of 5555887A by adding these together with the offset of 7A from the instruction. It then stores the content of D1 in that location. Neither the content of A3 nor that of D5 is changed by the instruction.



7 PROGRAM COUNTER RELATIVE ADDRESSING MODES IN THE MC68000

The PC-relative addressing mode is implemented in the MC68000 under two different names; "program counter with displacement" and "program counter with index." These two modes are similar to the two address register indirect modes: "address register indirect with displacement" and "address register indirect with index." The program counter is used instead of a specified address register as the base register.

The instruction `MOVE D1,2000H(PC)` would store the content of D1 into the memory location whose address is found by adding the offset value of 2000 hex to the content of the program counter.

A similar mode, sometimes referred to as *relative addressing*, is used only with certain branch instructions. As with other instructions using the PC-relative modes, the offset value must be calculated not from the location of the branch instruction itself, but from the beginning of the next instruction in sequence.

Including the displacement in the mnemonic form for a branch instruction would require the programmer to calculate the displacement value. To simplify the programming task the programmer will instead write a branch instruction in mnemonic form by *naming* the target destination address for the branch and writing that *name* instead of the displacement in the instruction. The displacement is calculated and included in the code at a later time when the program is translated into binary form (usually by a computer).

The example in Figure is an instruction to branch (unconditionally) to the instruction at the location named LOOP (BRA LOOP in mnemonic form). The instruction is in location 4568 and the op-code for BRA is 60. The displacement from the following instruction at 456A back to LOOP at 4560 is -0A, which is P8 in 2's complement form. Hence, the coding for this instruction is 60F8 and includes no extension words.

The processor executes the instruction by adding to the content of the program counter the displacement value of F8 specified in the instruction, sign-extended to FFFF8. This new address, which is loaded into the program counter (004560), is the location from which the next instruction will be fetched.

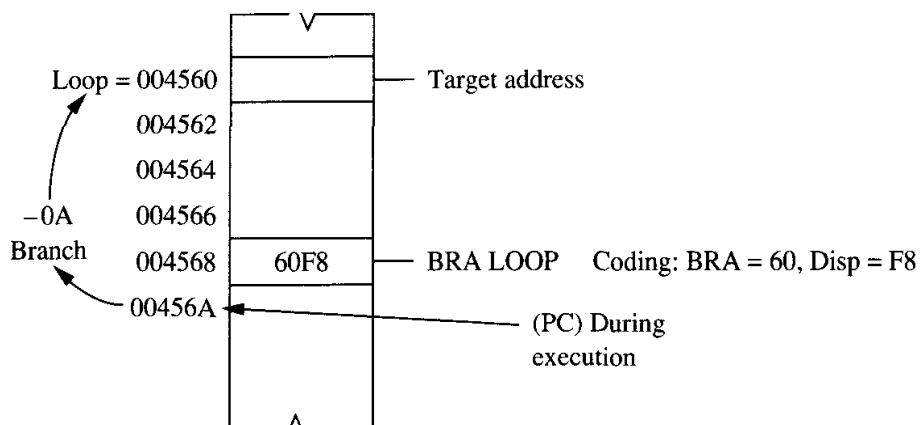


FIGURE (PC) Relative addressing in the MC68000: eight-bit displacement.

Notice also that the offset can range only from — 128 decimal to +127 decimal since that is the range of an eight-bit 2's complement number. If the distance to the target address is farther, then more bits must be used to specify the displacement.