INSTRUCTIONS AND ADDRESSING MODES

In its basic operation the central processing unit alternates between fetching instructions and executing instructions. The instructions cause the processor to manipulate the contents of specific programming model register- and of memory and I/O locations. Certain types of instructions. namely load/store, arithmetic logic, and input/output, must identify specific registers and/or locations for the processor in order that it may locate operands or store results property. Test branch instructions must also identify the branch target locations in the program. This chapter examines the various classes of instructions which processors must be able to execute.

1 TYPES OF INSTRUCTIONS

The computer can do certain basic types of operations: load/store, arithmetic logic, test/branch, and input/output. Although the instructions which direct these operations in the central processing unit vary widely from processor to processor, certain common characteristics are found in all processor instruction sets.

1.1 Load/Store Instructions

Load/store operations are those which move data between a register in the processor and a memory location (or another register). They are often collectively referred to as *data movement* instructions. The two terms *load* and *store* are often confused and should be carefully distinguished from each other. Figure 5.1 illustrates the difference.

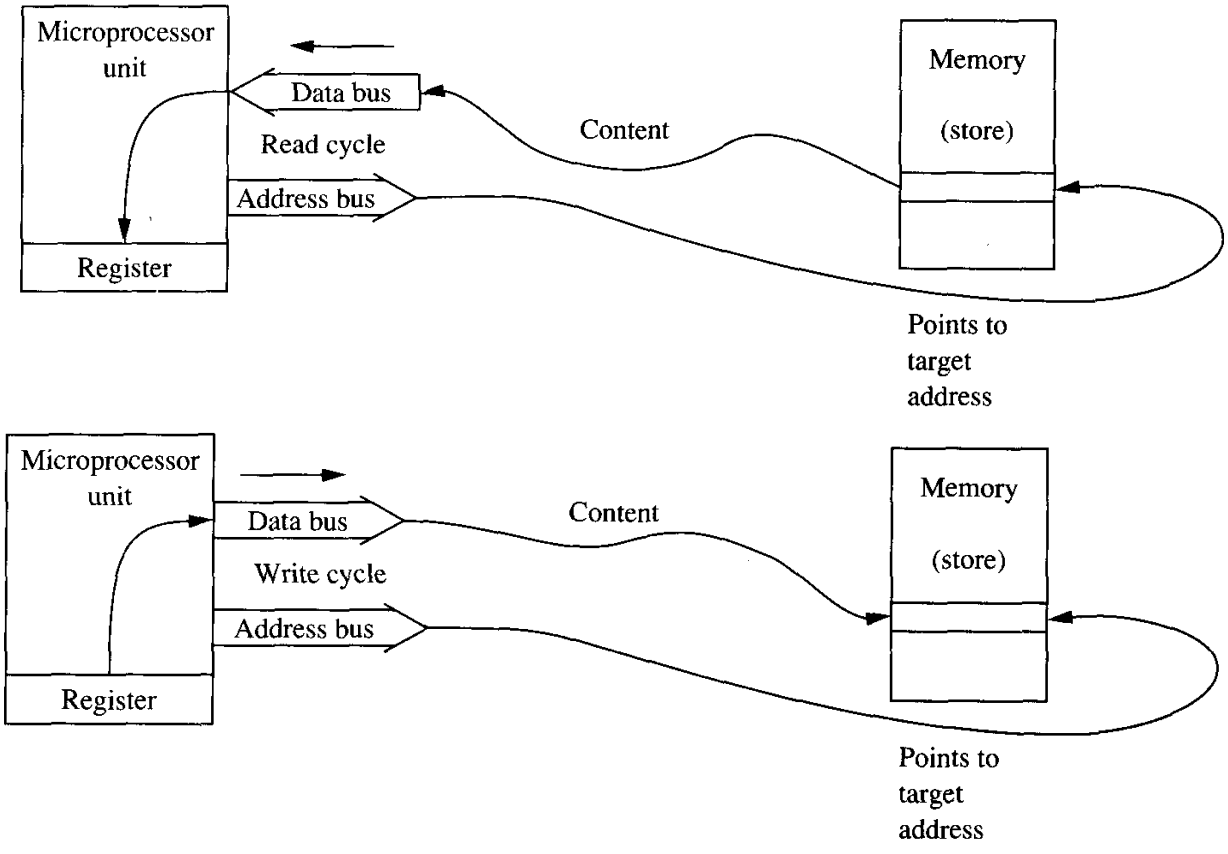


FIGURE 5.1Load and store operations: *(a)* load, *(b)* store.

One of the universal characteristics of instruction execution is that the source does not get changed, only the destination. Thus, in a load operation the processor copies data into a register, and in a store operation the processor copies data from a register into a memory location.

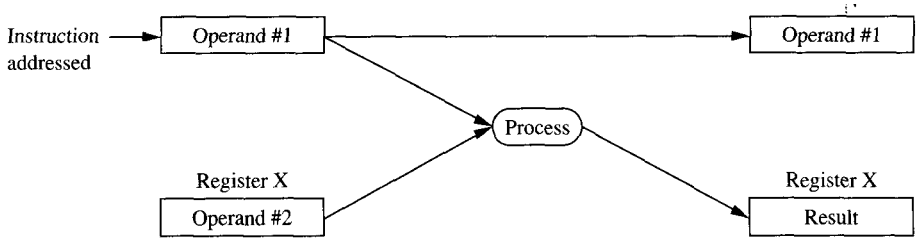
Examples of load/store instructions in the MC6809 are LDA, STA, LDB, STB, LDD, STD, and so forth. Examples of load/store instructions in the MC68000 are MOVE (for either direction of data movement), MOVEM (for moving the content of several registers into or out of memory with a single instruction), and EXG (for exchanging register contents). These instructions and others like them must include information identifying the source and destination.

5.1.2 Arithmetic/Logic Instructions

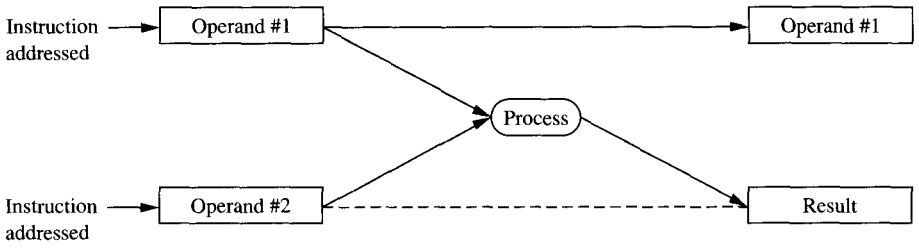
Arithmetic/logic operations provide the primary data-processing capabilities of the computer. The minimum arithmetic operations required are those of addition and subtraction.

All processor, implement the basic logic operations of AND, OR, and exclusive-OR on a bit-by-bit basis between two operands.

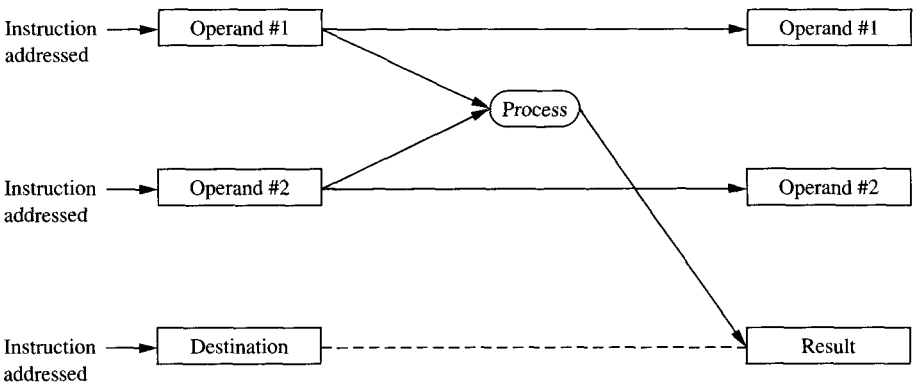
With operations requiring two operands, smaller microprocessors usually require that one of the operands must be in a destination register initially, while the other may be in another register or in memory. Their instructions may identify at most a single operand in memory, as shown in Figure *5.2a.* They support such operations as adding or ANDing to processor registers as illustrated but not to memory locations.



(*a*)



(*b*)



(*c*)

FIGURE 5.2 Instruction formats: *(a)* single address, *(b)* double address, (c) triple address.

Some examples of two-operand arithmetic/logic instructions in the MC6809 are ADDB. SUBA. SUBD, ANDA, and EORA (exclusive-OR). Each of these requires one operand to be in an accumulator

Some two-operand arithmetic/logic instructions in the MC68000 instruction set are ADD, AND, OR, SUB, and EOR. All of the two-operand instructions in the MC68000 must specify the source and the destination. With a few exceptions, these may both be registers or memory locations or one of each.

Among the single-operand arithmetic/logic operations are increment and decrement operations, the 1's and 2's complement, and various shift and *rotate* operations. These operate directly upon the specified operand to modify it. Thus, the operand location is both the source and the destination for the operation.

Examples of one-operand instructions in the MC6809 include COMA (complement A), ASLB (arithmetic shift left B), INC (increment a memory location whose address is identified in the instruction), and NEGA (2's complement or negate A). The MC68000 includes the instructions CLR (clear), NEG (change sign of), NOT (complement), and EXT (extend the sign bit to double the number of bits in an operand).

1.3 Test/Branch Instructions

Test/branch operations provide the decision-making capabilities so important in computer programs. The test must often be performed on the result of some prior arithmetic/logic operation. It determines whether the result was zero or nonzero, positive or negative, or some other binary choice involving bits in the condition code register. During the execution of a test/branch instruction the processor examines the appropriate condition code bits to determine the result of the test. Depending upon the result, the processor may or may not branch to a remote location for the next instruction.

Examples of test/branch instructions are BCC (branch if carry is cleared), BEQ (branch if equal to 0), BMI (branch if minus), and BRA (branch always).

2 ADDRESSING MODES

Most of the instructions must refer to the address or content of a specific memory location. These so-called *memory reference instructions* must somehow identify the address of the location as a part of the instruction encoding. The manner in which this *target address* or *effective address* is identified within the instruction is called the *addressing mode.*

This section describes the more common addressing modes used in microprocessors.

2.1 Direct Addressing

When the instruction explicitly states the location of an operand or a destination (either in memory or in a processor register), the addressing mode is known as *direct addressing.* The effective address itself is included in the subsequent words of the instruction *(post-words).*

Two subclassifications within the direct addressing mode are often recognized. When the location is in memory the mode may be referred to as *absolute addressing.* When the location is a processor register it may be referred to as *register direct addressing.* Figure 5.3 illustrates the two

modes. In part *a* the instruction specifies the address of the operand; in part *b* the instruction specifies the register containing the operand.

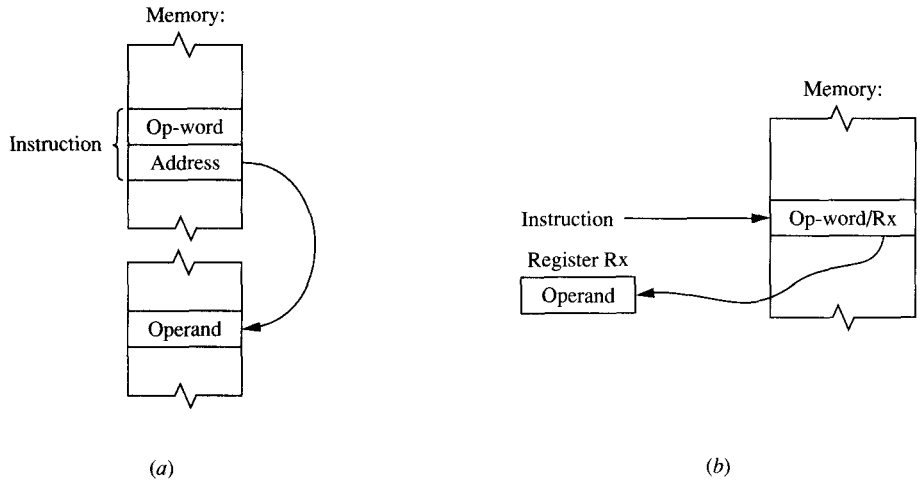
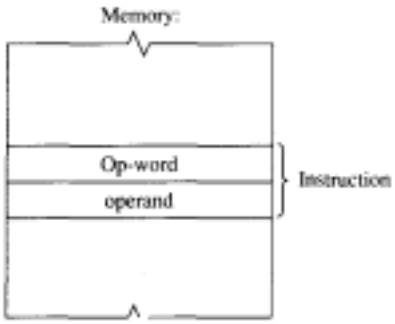


FIGURE 5.3 Direct addressing: (a) absolute addressing, *(b)* register direct addressing.
(LDA 100BH, ADDA 100CH, STA 100DH

Direct addressing is used when the memory address or the selected register is to be fixed in the program.

2.2 Immediate Addressing

In many cases an instruction requires a constant quantity, a bit pattern which will never change no matter when or how often the instruction is executed. This mode of including a bit pattern as a part of an instruction is called the *immediate addressing* mode. The op-word for the instruction includes a group of bits which identifies this mode of addressing, and the post-words include the bit pattern itself. Since the instruction is located in program memory the constant itself is also in program memory.



The immediate addressing mode the instruction does not state explicitly the location of the operand; rather, it explicitly states the operand itself. The example in Figure 5.4 illustrates this mode. Note that the operand becomes an integral part of the instruction.

FIGURE 5.4 Immediate addressing.

Immediate addressing is used when a particular constant value is to be fixed within the program itself. The value is found in memory "immediately" after the instruction code word and may never change at any time.

2.3 Indirect Addressing

In the *indirect addressing* mode the instruction tells the processor neither the address of the operand nor the operand itself. Instead, it tells the processor where to go to find the address of the operand. The instruction may explicitly state either the address of a location in memory or the name

of a processor register, but the binary number which is found there is not the operand. Instead, it is the effective address, the address of a location in memory to which the processor must go to find the operand. The result is that the processor must take one extra step in order to locate the operand.

The op-word for the instruction includes a group of bits which identifies this mode of addressing, and the (indirect) address is specified in one or more additional post-words. If the instruction names a processor register as the source of the effective address, then the register identification number may fit into the op-word itself.

Indirect addressing is used when a program must operate upon different data values under different circumstances.

Figure 5.6 compares and contrasts the first three addressing modes. Note that in the immediate mode the instruction includes the operand, in the direct mode it includes the address of the operand, and in the indirect mode it includes the address of the address of the operand.

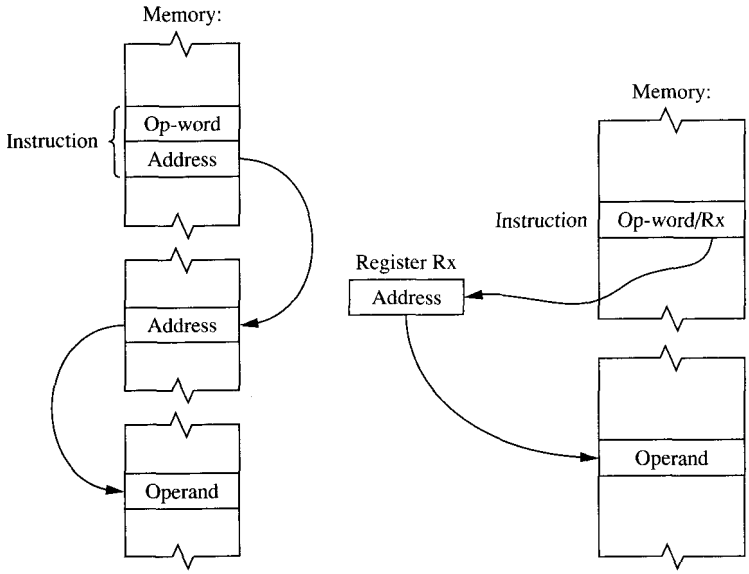


FIGURE 5.5 Indirect addressing: (a) memory indirect, *(b)* register indirect.

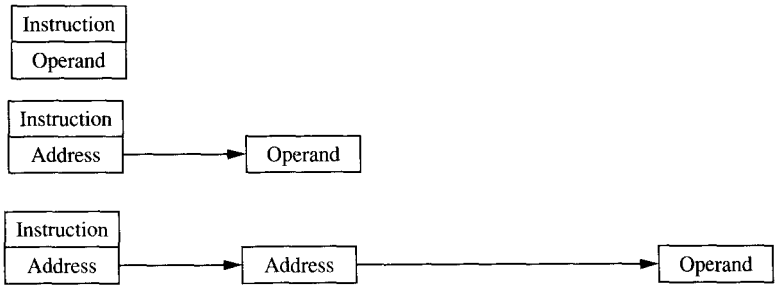


FIGURE 5.6 Comparison of immediate, direct, and indirect addressing.

2.4 Multi-Component Addressing Modes

Each of the following related addressing modes requires that the processor assemble two or more components together during the execution of the program in order to create the effective address. In each case the effective address itself is that of a memory location. However, at least one of the components is found in a processor register.

Instructions which use *indexed addressing* specify two registers, often by coding within the op-word itself and known as "indexed addressing". During program execution the processor temporarily

adds the contents of these registers to generate the effective address. One of the registers is an address register and it is said to hold the *base address.* The other is commonly a data register - the *displacement* or *index* register.

Based addressing is a similar mode wherein the instruction specifies an address register and a fixed constant (an *offset* or *displacement).* The register designation often fits within the op-word and the offset usually requires post-words. In this mode the content of the register is the base and the constant is the displacement. During execution the processor adds the constant and the value in the register to generate the effective address. This addressing mode is also known as "relative based indexed" mode

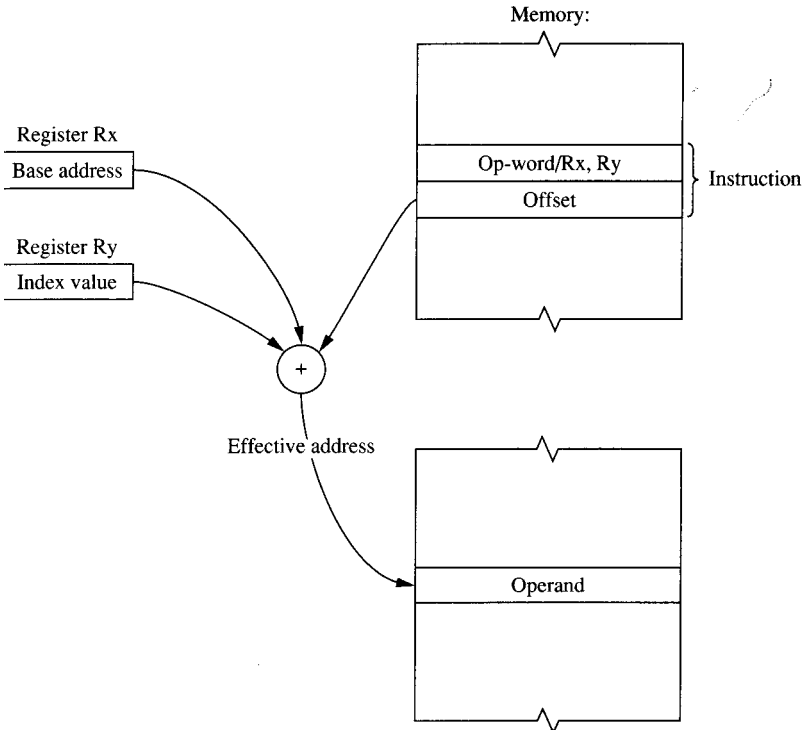


FIGURE 5.7 "Indexed/based/offset" addressing.

The relative addressing mode permits the writing of "position-independent code," programs which will be properly executed by the processor regardless of where they are located in memory. The entire program (together with any necessary data) may be picked up from one region of memory and moved to another with no adverse effect. In order to be location-independent a program may not refer to any specific location by address. All references to memory must be through the use of relative addressing. Examples will be shown in a later chapter.

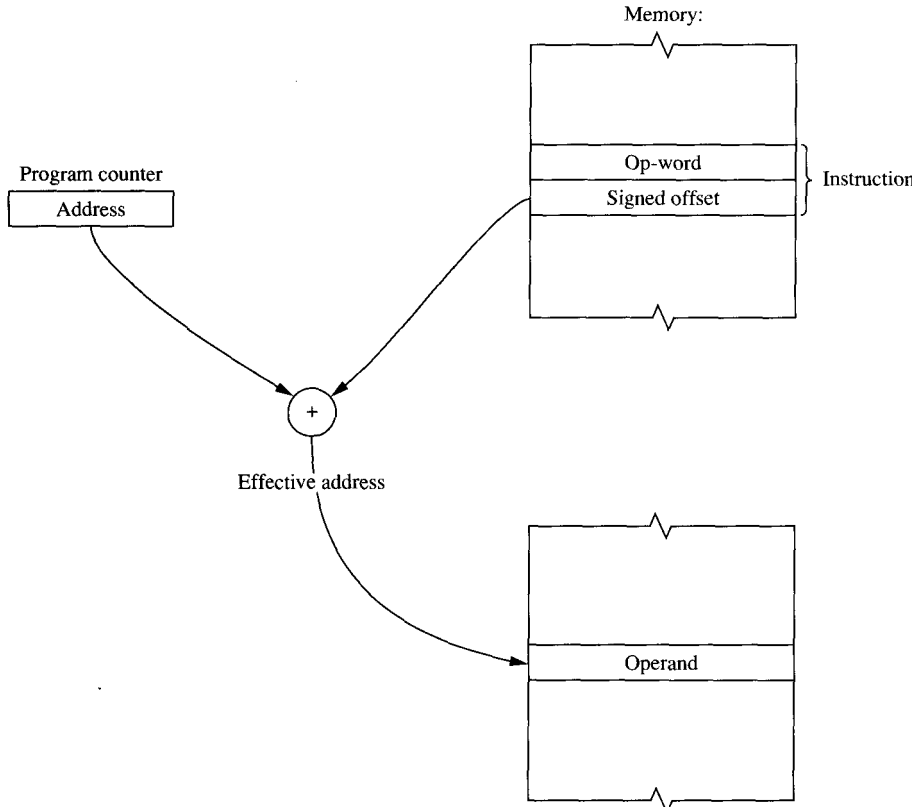


FIGURE 5.8 (PC) Relative addressing.

2.5 Implied or Inherent Addressing

Certain instructions allow no choice of register or location but always cause the processor to refer to the same registers. One example is the multiply instruction in the MC6809 which always assumes that the operands are in accumulators A and B. Examples in the MC6809 are ABX (add B to X), CLRA (clear A), and TSTA (test A).

3 THE MC6809 ADDRESSING MODES

This section describes in detail the addressing modes used in the Motorola MC6809 microprocessor, their formats within program memory, and how they are specified in mnemonic form. The terminology is that used in the manufacturer's literature.

The MC6809 includes five addressing modes: extended, immediate, direct, relative, and indexed. In addition, the indexed mode has a large variety of options. Because of the large number of instructions and addressing modes, the instructions vary in length from one to five bytes.

The basic addressing modes are illustrated in Figure 5.10

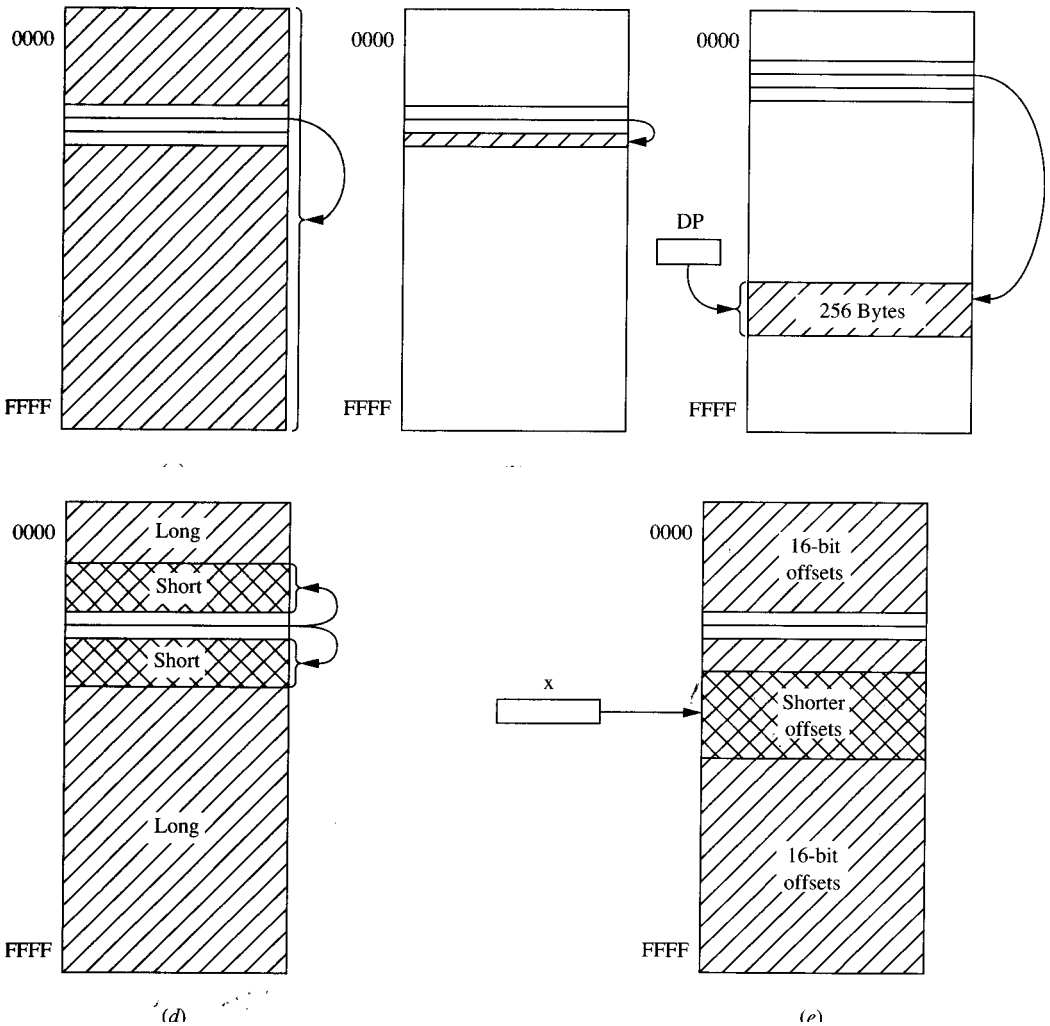


FIGURE 5.10 MC6809 addressing modes: (a) extended, (b) immediate, (c) direct, (d) relative (branches only), (e) indexed.

Auto Increment/Decrement Options in the MC6809

While repeating a program loop the processor must often change a pointer to an adjacent address in memory on each pass through the loop. It may be necessary to increment or decrement the value in the pointer, depending upon the direction in which the array is being scanned.

Two examples of auto increment/decrement options are shown in Figure 5.18. Part *a* is the instruction LDA ,X+, load A indexed with X post-inc by one. Figure 5.18b is the instruction STY ,- -U. When it is executed the processor will first decrement the index register U by 2. Following that, it will store the upper half of the two-byte register Y into the new location indicated by register U and the lower half into the next subsequent location.

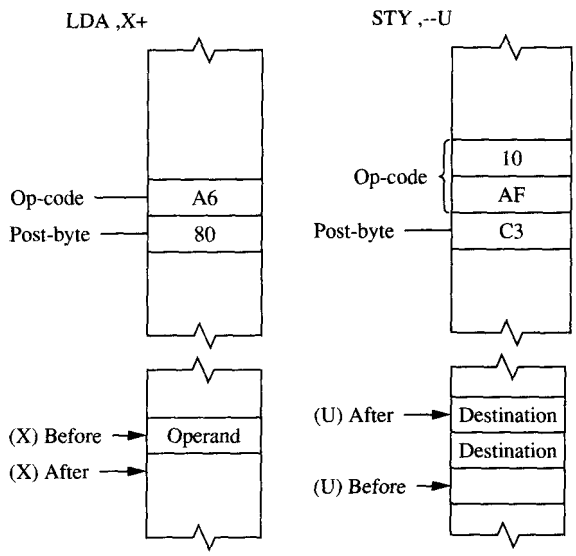


FIGURE 5.18 MC6809 indexed addressing, auto increment/decrement options: (a) post-increment, *(b)* pre-decrement.

Pre-decrementing by 2 opens up two new slots in memory where the double-byte value in Y can be stored in accordance with the instruction.

As a consequence of the increment/decrement option, not only does the instruction accomplish its primary task of manipulating the targeted operands, it also modifies the content of the selected index register.