7. Microprocessor instruction set. Instruction format. Opcode and operand fields.
8. Addressing modes according to the way of the operand address forming.

When the 8088 executes an instruction, it performs the specified function on data. These data called operands, reside in some of the following objects:
– in a part of the instruction;
– in one of the internal registers of the microprocessor;
– stored at an address in memory.

Next in the text the MOV (move data) instruction is used to describe the data-addressing modes. The MOV instruction transfers bytes or words of data between registers or between registers and memory in the 8086. In describing the program memory addressing modes, the CALL and JMP instructions show how to modify the flow of the program.

The data addressing modes include register, immediate, direct, register indirect, base plus index, register relative, and base relative plus index in the 8086 microprocessors. The program memory addressing modes include program relative, direct and indirect. The operation of the stack memory is explained for the specific instructions POP and PUSH.

Because the MOV instruction is common and flexible, it provides a basis for the explanation of the data-addressing modes. Figure 3–1 illustrates the MOV instruction and defines the direction of data flow. The **source** is to the right and the **destination** is to the left, next to the opcode MOV. (An **opcode** or operation code tells the microprocessor which operation to perform.) This direction of flow, which is applied to all instructions, initially seems awkward. We naturally assume that things move from left to right, where as here they move from right to left. Notice that a *comma* always separates the destination from the source in an instruction. Also note that memory-to-memory transfers are not allowed by any instruction except for the MOVS instruction.

In Figure 3–1, the MOV AX,BX instruction transfers the word contents of the source register (BX) into the destination register (AX). The source never changes, but the destination almost always changes.[1] It is essential to remember that a MOV instruction always *copies* the source data into the destination. The MOV never actually picks up the data and moves it. Also note that the flag register remains unaffected by most data transfer instructions.

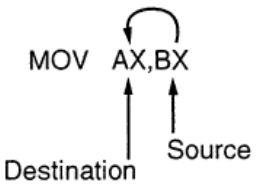


FIGURE 3–1 The MOV instruction showing the source, destination, and direction of data flow

Fig. 1 shows all possible variation of the data-addressing modes using the MOV instruction. It helps to understand how each data-addressing mode is formulated with MOV instruction and also serves as a reference. These are the same addressing modes found with all version of the Intel microprocessors. The data-addressing modes are:

Register addressing — Transfers a copy of a byte or word from the source register or memory location to the destination register or memory location. (Example: the MOV CX,DX instruction copies the word-sized contents of register DX into register CX.) In the 80386 and above, a doubleword can be transferred from the source register or memory location to the destination register or memory location. (Example: the MOV ECX,EDX instruction copies the doubleword-sized contents of register EDX into register ECX.)

Immediate addressing

Transfers the source-immediate byte or word of data into the destination register or memory location. (Example: the MOV AL,22H instruction copies a byte-sized 22H into register AL.) In the 80386 and above, a doubleword of immediate data can be transferred into a register or memory location. (Example: the MOV EBX,12345678H instruction copies a doubleword-sized 12345678H into the 32-bit wide EBX register.)

Direct addressing

Moves a byte or word between a memory location and a register. The instruction set does not support a memory-to-memory transfer, except for the MOVS instruction. (Example: the MOV CX,LIST instruction copies the word-sized contents of memory location LIST into register CX.) In the 80386 and above, a doubleword-sized memory location can also be addressed. (Example: the MOV ESI,LIST instruction copies a 32-bit number, stored in four consecutive bytes of memory, from location LIST into register ESI.)

Register indirect addressing

Transfers a byte or word between a register and a memory location addressed by an index or base register. The index and base registers are BP, BX, DI, and SI. (Example: the MOV AX,[BX] instruction copies the word-sized data from the data segment offset address indexed by BX into register AX.) In the 80386 and above, a byte, word, or doubleword is transferred between a register and a memory location addressed by any register: EAX, EBX, ECX, EDX, EBP, EDI, or ESI. (Example: the MOV AL,[ECX] instruction loads AL from the data segment offset address selected by the contents of ECX.)

Base-plus-index addressing

Transfers a byte or word between a register and the memory location addressed by a base register (BP or BX) plus an index register (DI or SI). (Example: the MOV [BX+DI],CL instruction copies the byte-sized contents of register CL into the data segment memory location addressed by BX plus DI.) In the 80386 and above, any register EAX, EBX, ECX, EDX, EBP, EDI, or ESI may be combined to generate the memory address. (Example: the MOV [EAX+EBX],CL instruction copies the byte-sized contents of register CL into the data segment memory location addressed by EAX plus EBX.)

Register relative addressing

Moves a byte or word between a register and the memory location addressed by an index or base register plus a displacement. (Example: MOV AX,[BX+4] or MOV AX,ARRAY[BX]. The first instruction loads AX from the data segment address formed by BX plus 4. The second instruction loads AX from the data segment memory location in ARRAY plus the contents of BX.) The 80386 and above use any register to address memory. (Example: MOV AX,[ECX+4] or MOV AX,ARRAY[EBX]. The first instruction loads AX from the data segment address formed by ECX plus 4. The second instruction loads AX from the data segment memory location ARRAY plus the contents of EBX.)

Base relative-plus-index addressing Transfers a byte or word between a register and the memory location addressed by a base and an index register plus a displacement. (Example: MOV AX,ARRAY[BX+DI] or MOV AX,[BX+DI+4]. These instructions both load AX from a data segment memory location. The first instruction uses an address formed by adding ARRAY, BX, and DI; the second, by adding BX, DI, and 4.) (An 80386 and above example: MOV EAX,ARRAY[EBX+ECX] loads EAX from the data segment memory location accessed by the sum of ARRAY, EBX, and ECX.)

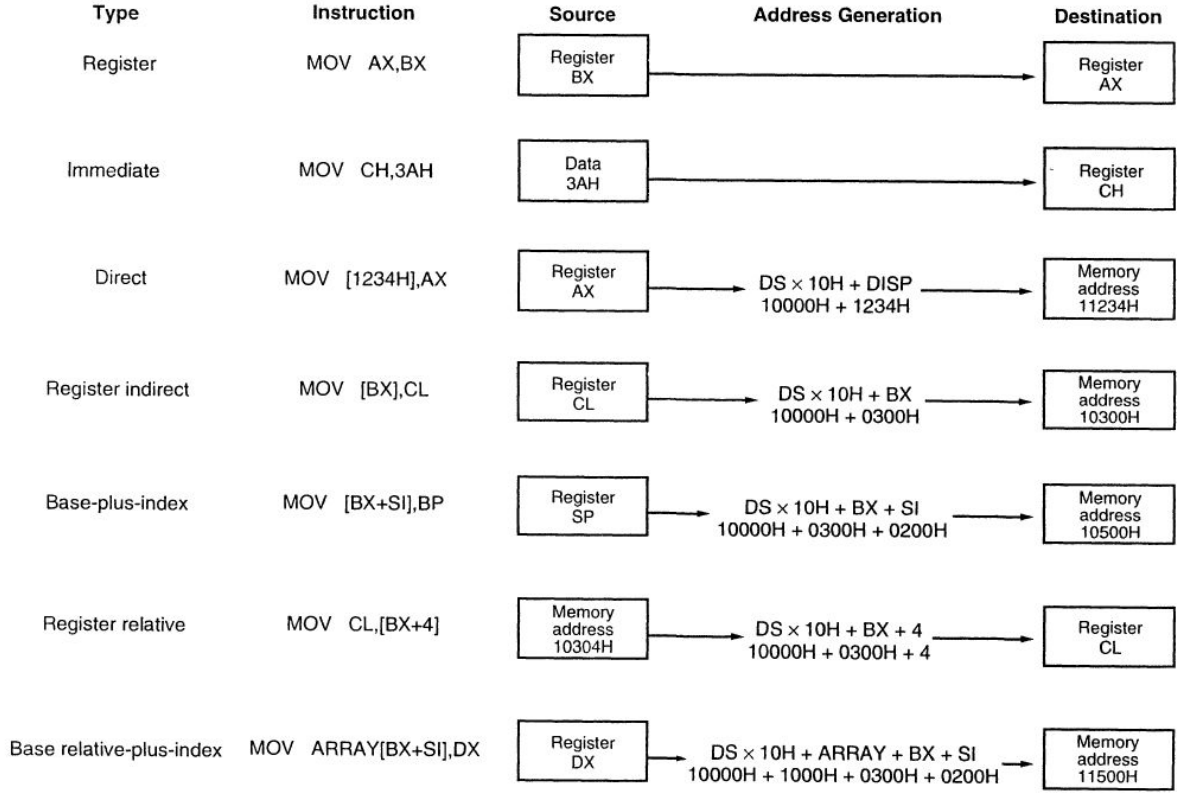


Fig. 1 8086/8088 Addressing modes

Register Addressing

Register addressing is the most common form of data addressing and, once the register names are learned, is the easiest to apply. The microprocessor contains the following 8-bit registers used with register addressing: AH, AL, BH, BL, CH, CL, DH, and DL. Also present are the following 16-bit registers: AX, BX, CX, DX, SP, BP, SI, and DI. In the 80386 and above, the extended 32-bit registers are EAX, EBX, ECX, EDX, ESP, EBP, EDI, and ESI. With register addressing, some MOV instructions and the PUSH and POP instructions also use the 16-bit segment registers (CS, ES, DS, SS, FS, and GS). It is important for instructions to use registers that are the same size. *Never* mix an 8-bit register with a 16-bit register, an 8-bit register with a 32-bit register, or a 16-bit register with 32-bit register, because this is not allowed by the microprocessor and results in an error when assembled. This is even true when a MOV AX,AL or a MOV EAX,AL instruction may seem to make sense. Of course, the MOV AX,AL or MOV EAX,AL instructions are *not* allowed, because these registers are of different sizes. Note that a few instructions, such as SHL DX,CL, are exceptions to this rule, as indicated in later chapters. It is also important to note that *none* of the MOV instructions affect the flag bits.

Table 3–1 shows many variations of register move instructions. It is impossible to show all of the many possible combinations. For example, just the 8-bit subset of the MOV instruction has 64 different variations. A segment-to-segment register MOV instruction is virtually the only type of register MOV instruction *not* allowed. Also note that the code segment register may not be changed by a MOV instruction, because the address of the next instruction is found in both IP/EIP and CS. If only CS were changed, the address of the next instruction would be unpredictable. Therefore, changing the CS register with a MOV instruction is not allowed.

TABLE 3–1 Examples of the register-addressed instructions

Assembly Language	*Size*	*Operation*
MOV AL,BL	8-bits	Copies BL into AL
MOV CH,CL	8-bits	Copies CL into CH
MOV AX,CX	16-bits	Copies CX into AX
MOV SP,BP	16-bits	Copies BP into SP
MOV DS,AX	16-bits	Copies AX into DS
MOV SI,DI	16-bits	Copies DI into SI
MOV BX,ES	16-bits	Copies ES into BX
MOV ECX,EBX	32-bits	Copies EBX into ECX
MOV ESP,EDX	32-bits	Copies EDX into ESP
MOV ES,DS	—	Not allowed (segment-to-segment)
MOV BL,DX	—	Not allowed (mixed sizes)
MOV CS,AX	—	Not allowed (the code segment register may not be the destination register)

Figure 3–3 shows the operation of the MOV BX,CX instruction. Note that the source register's contents do not change, but the destination register's contents do change. This instruction moves *(copies)* a 1234H from register CX into register BX. This *erases* the old contents (76AFH) of register BX, but the contents of CX remain unchanged. The contents of the destination register or destination memory location change for all instructions except the CMP and TEST instructions. Note that the MOV BX,CX instruction does not affect the leftmost 16-bits of register EBX.

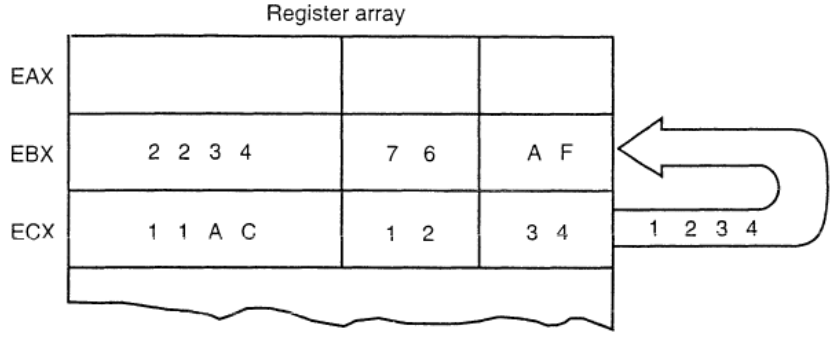


FIGURE 3–3 The effect of executing the MOV BX, CX instruction at the point just before the BX register changes. Note that only the rightmost 16-bits of register EBX change.

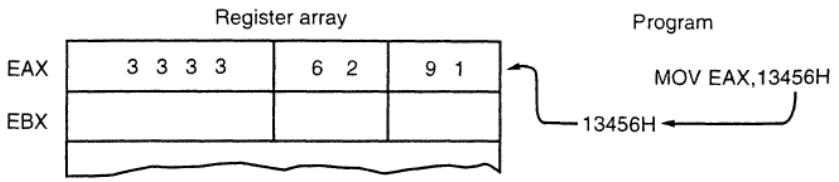


FIGURE 3–4 The operation of the MOV EAX,3456H instruction. This instruction copies the immediate data (13456H) into EAX.

Immediate Addressing

Another data-addressing mode is immediate addressing. The term *immediate* implies that the data immediately follow the hexadecimal opcode in the memory. Also note that immediate data are **constant data,** while the data transferred from a register are **variable data.** Immediate addressing operates upon a byte or word of data. In the 80386 through the Pentium Pro microprocessors immediate addressing also operates on doubleword data. The MOV immediate instruction transfers a copy of the immediate data into a register or a memory location. Figure 3–4 shows the operation of a MOV EAX,13456H instruction. This instruction copies the 13456H from the instruction, located in the memory immediately following the hexadecimal opcode, into register EAX. As with the MOV instruction illustrated in Figure 3–3, the source data overwrite the destination data.

Direct Addressing. Direct addressing, with a MOV instruction, transfers data between a memory location, located within the data segment, and the AL (8-bit), AX (16-bit), or EAX (32-bit) register. A MOV instruction using this type of addressing is usually a 3-byte long instruction. (In the 80386 and above, a register size prefix may appear before the instruction, causing it to exceed three bytes in length.)

The MOV AL,DATA instruction, as represented by most assemblers, loads AL from data segment memory location DATA (1234H). Memory location DATA is a symbolic memory location, while the 1234H is the actual hexadecimal location. With many assemblers, this instruction is represented as a MOV AL,[1234H] instruction.[7] The [1234H] is an absolute memory location that is not allowed by all assembler programs. Note that this may need to be formed as MOV AL,DS:[1234H] with some assemblers, to show that the address is in the data segment. Figure 3–5 shows how this instruction transfers a copy of the byte-sized contents of memory location 11234H into AL. The effective address is formed by adding 1234H (the offset address) to 10000H (the data segment address) in a system operating in the real mode.

Table 3–3 lists the three direct addressed instructions. These instructions often appear in programs, so Intel decided to make them special 3-byte long instructions to reduce the length of programs. All other instructions that move data from a memory location to a register, called *displacement addressed instructions,* require four or more bytes of memory for storage in a program.

TABLE 3–3 Direct addressed instructions using EAX, AX and AL

Assembly Language	*Size*	*Operation*
MOV AL,NUMBER	8-bits	Copies the byte contents of data segment memory location NUMBER into AL
MOV AX,COW	16-bits	Copies the word contents of data segment memory location COW into AX
MOV EAX,WATER*	32-bits	Copies the doubleword contents of memory location WATER into EAX
MOV NEWS,AL	8-bits	Copies AL into data segment memory location NEWS
MOV THERE,AX	16-bits	Copies AX into data segment memory location THERE
MOV HOME,EAX*	32-bits	Copies EAX into data segment memory location HOME

Register Indirect Addressing

Register indirect addressing allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI, and SI. For example, if register BX contains a 1000H and the MOV AX,[BX] instruction executes, the word contents of data segment

offset address 1000H is copied into register AX. If the microprocessor is operated in the real mode and DS = 0100H, this instruction addresses a word stored at memory bytes 2000H and 2001H and transfers it into register AX (see Figure 3–6). Note that the contents of 2000H are moved into AL and the contents of 2001H are moved into AH. The [ ] symbols denote indirect addressing in assembly language.

TABLE 3–4 Examples of direct data addressing using a displacement

Assembly Language	*Size*	*Operation*
MOV CH,DOG	8-bits	Copies the byte contents of data segment memory location DOG into CH
MOV CH,[1000H]	8-bits	Copies the byte contents of data segment offset address 1000H into CH
MOV ES,DATA6	16-bits	Copies the word contents of data segment memory location DATA6 into ES
MOV DATA7,BP	16-bits	Copies BP into data segment memory location DATA7
MOV NUMBER,SP	16-bits	Copies SP into data segment memory location NUMBER
MOV DATA1,EAX	32-bits	Copies EAX into data segment memory location DATA1
MOV EDI,SUM1	32-bits	Copies the doubleword contents of data segment memory location SUM1 into EDI

The **data segment** is used by default with register indirect addressing or any other addressing mode that uses BX, DI, or SI to address memory. If register BP addresses memory, the **stack segment** is used by default. These are considered the default settings for these four index and base registers.

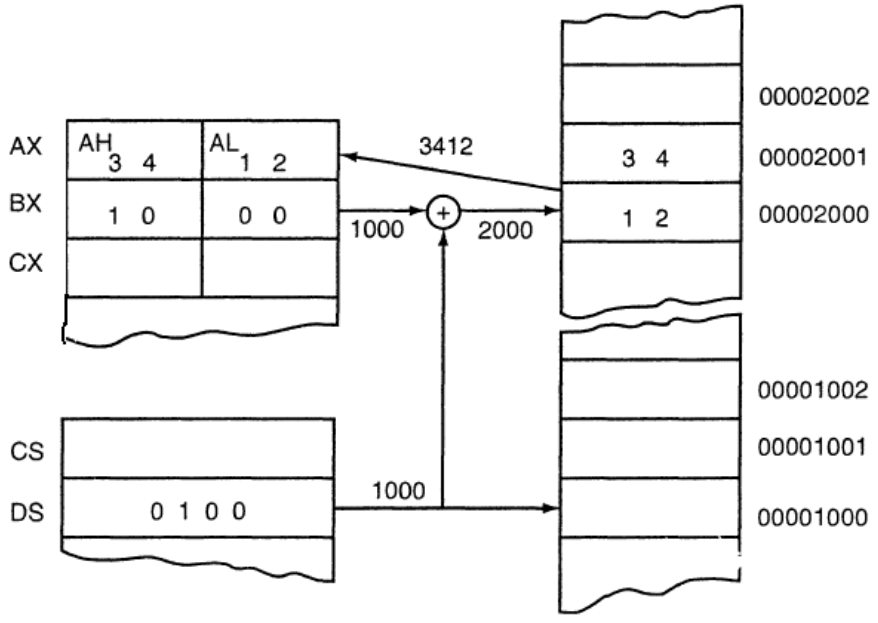


FIGURE 3–6 The operation of the MOV AX,[BX] instruction when BX = 1000H and DS = 0100H. Note that this instruction is shown after the contents of memory are transferred to AX.

TABLE 3–5 Example of register indirect addressing

Assembly Language	*Size*	*Operation*
MOV CX,[BX]	16-bits	Copies the word contents of the data segment memory location address by BX into CX
MOV [BP],DL*	8-bits	Copies DL into the stack segment memory location addressed by BP
MOV [DI],BH	8-bits	Copies BH into the data segment memory location addressed by DI
MOV [DI],[BX]	—	Memory-to-memory moves are not allowed except with string instructions

Indirect addressing often allows a program to refer to tabular data located in the memory system. For example, suppose that you must create a table of information that contains 50 samples taken from memory location 0000:046C. Location 0000:046C contains a counter that is maintained by the personal computer's real-time clock. Figure 3–7 shows the table and the BX register used to address each location in the table sequentially. To accomplish this task, load the starting location of the table into the BX register with a MOV immediate instruction, After initializing the starting address of the table, use register indirect addressing to store the 50 samples sequentially.

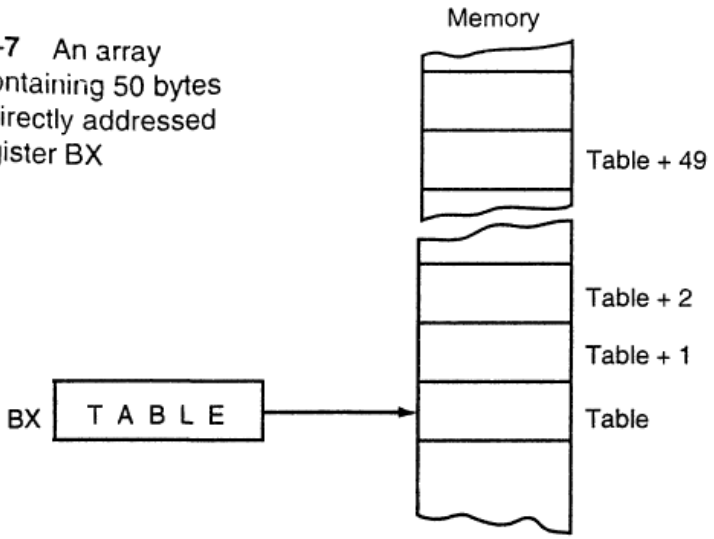


FIGURE 3–7 An array (TABLE) containing 50 bytes that are indirectly addressed through register BX

Base-Plus-Index Addressing

Base-plus-index addressing is similar to indirect addressing because it indirectly addresses memory data. In the 8086 through the 80286, this type of addressing uses one base register (BP or BX) and one index register (DI or SI) to indirectly address memory. The base register often holds the beginning location of a memory array, while the index register holds the relative position of an element in the array. Remember that whenever BP addresses memory data, both the stack segment register and BP generate the effective address.

Locating Data with Base-plus-index Addressing. Figure 3–8 shows how data are addressed by the MOV DX,[BX+DI] instruction when the microprocessor operates in the real mode. In this example, BX = 1000H, DI = 0010H, and DS = 0100H, which translate into memory address 02010H. This instruction transfers a copy of the word from location 02010H into the DX register. Table 3–6 lists some instructions used for base-plus-index addressing. Note that the Intel assembler requires that this addressing mode appear as [BX][DI] instead of [BX+DI]. The MOV DX,[BX+DI] instruction is MOV DX,[BX][DI] for a program written for the Intel ASM assembler. This text uses the first form in all example programs, but the second form can be used in many assemblers, including MASM from Microsoft.

Locating Array Data Using Base-plus-index Addressing. A major use of the base-plus-index addressing mode is to address elements in a memory array. Suppose that the elements in an array located in the data segment at memory location ARRAY must be accessed. To accomplish this, load the BX register (base) with the beginning address of the array and the DI register (index) with the element number to be accessed. Figure 3–9 shows the use of BX and DI to access an element in an array of data.

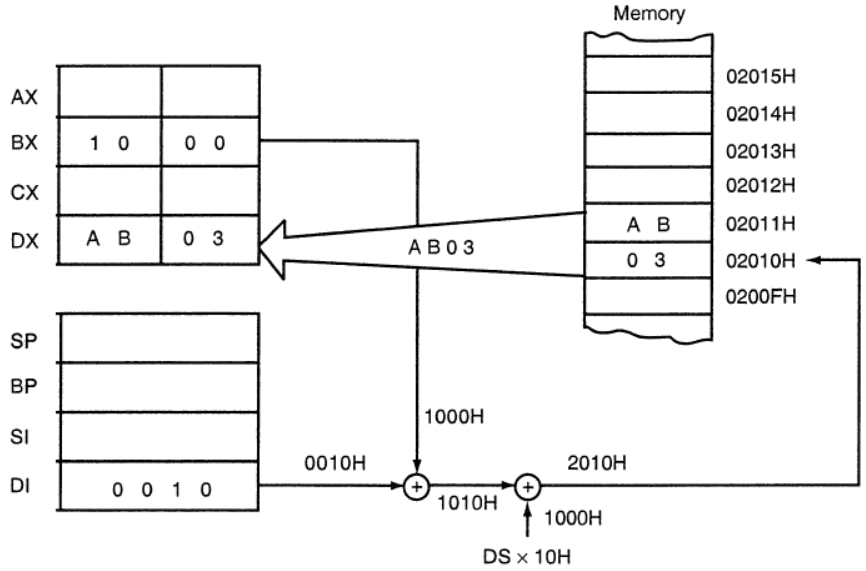


FIGURE 3–8 An example showing how the base-plus-index addressing mode functions for the MOV DX,[BX+DI] instruction. Notice that memory address 02010H is accessed because DS = 0100H, BX = 100H, and DI = 0010H.

TABLE 3–6 Examples of base-plus-index addressing

Assembly Language	*Size*	*Operation*
MOV CX,[BX+DI]	16-bits	Copies the word contents of the data segment memory location address by BX plus DI into CX
MOV CH,[BP+SI]	8-bits	Copies the byte contents of the stack segment memory location addressed by BP plus SI into CH
MOV [BX+SI],SP	16-bits	Copies SP into the data segment memory location addresses by BX plus SI
MOV [BP+DI],AH	8-bits	Copies AH into the stack segment memory location addressed by BP plus DI

Register Relative Addressing

Register relative addressing is similar to base-plus-index addressing and displacement addressing. In register relative addressing, the data in a segment of memory are addressed by adding the displacement to the contents of a base or an index register (BP, BX, DI, or SI). Figure 3–10 shows the operation of the MOV AX,[BX+1000H] instruction. In this example, BX = 0100H and DS = 0200H, so the address generated is the sum of DS × 10H, BX, and the displacement of 1000H or 03100H. Remember that BX, DI, or SI address the data segment and BP addresses the stack segment. In the 80386 and above, the displacement can be a 32-bit number and the register can be any 32-bit register except the ESP register. Remember that the size of a real mode segment is 64K bytes long. Table 3–7 lists a few instructions that use register relative addressing.

TABLE 3–7 Examples of register relative addressing

Assembly Language	*Size*	*Operation*
MOV AX,[DI+100H]	16-bits	Copies the word contents of the data segment memory location addressed by DI plus 100H into AX
MOV ARRAY[SI],BL	8-bits	Copies BL into the data segment memory location addressed by ARRAY plus SI
MOV LIST[SI+2],CL	8-bits	Copies CL into the data segment memory location addressed by sum of LIST, SI, and 2
MOV DI,SET_IT[BX]	16-bits	Copies the word contents of the data segment memory location addressed by the sum of SET_IT and BX into DI

The displacement can be a number added to the register within the [ ], as in MOV AL,[DI+2], or it can be a displacement subtracted from the register, as in MOV AL,[SI–1]. A displacement also can be an offset address appended to the front of the [ ], as in MOV AL,DATA[DI]. Both forms of displacements also can appear simultaneously, as in MOV AL,DATA[DI+3]. In all cases, both forms of the displacement add to the base or base and index register within the [ ]. In the 8086–80286 microprocessors, the value of the displacement is limited to a 16-bit signed number with a value ranging between +32,767 (7FFFH) and –32,768 (8000H);

Addressing Array Data with Register Relative Addressing. It is possible to address array data with register relative addressing such as one does with base-plus-index addressing. In Figure 3–11, register relative addressing is illustrated with the same example as for base-plus-index addressing. This shows how the displacement ARRAY adds to index register DI to generate a reference to an array element.

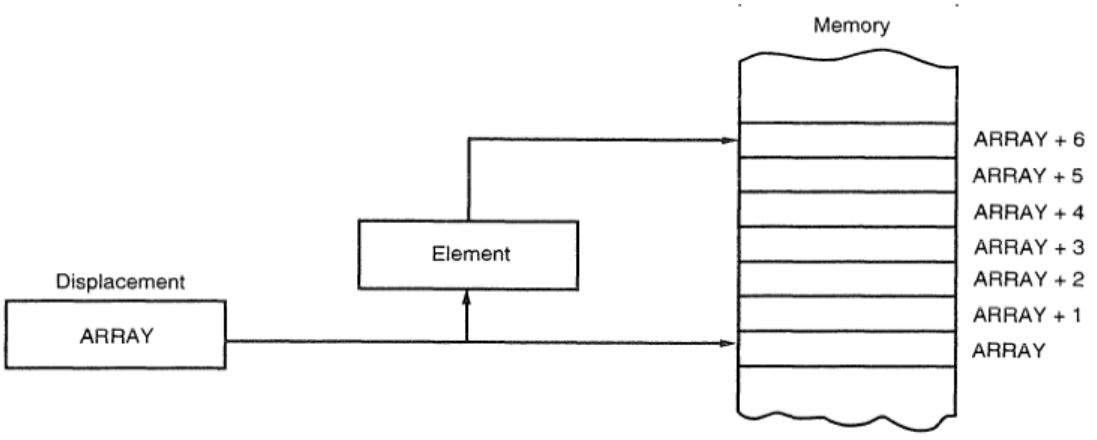


FIGURE 3–11 Register relative addressing used to address an element of ARRAY. The displacement addresses the start of ARRAY, and DI accesses an element.

Base Relative-Plus-Index Addressing

The base relative-plus-index addressing mode is similar to the base-plus-index addressing mode, but adds a displacement besides using a base register and an index register to form the memory address. This type of addressing mode often addresses a two-dimensional array of memory data.

Addressing Data with Base Relative-plus-index Addressing. Base relative-plus-index addressing is the least-used addressing mode. Figure 3–12 shows how data are referenced if the instruction executed by the microprocessor is a MOV AX,[BX+SI+100H]. The displacement of 100H adds to BX and SI to form the offset address within the data segment. Registers BX = 0020H, SI = 0010H, and DS = 1000H, so the effective address for this instruction is 10130H—the sum of these registers plus a displacement of 100H. This addressing mode is too complex for frequent use in a program. Some typical instructions using base relative-plus-index addressing appear in Table 3–8. Note that with the 80386 and above, the effective address is generated by the sum of two 32-bit registers plus a 32-bit displacement.

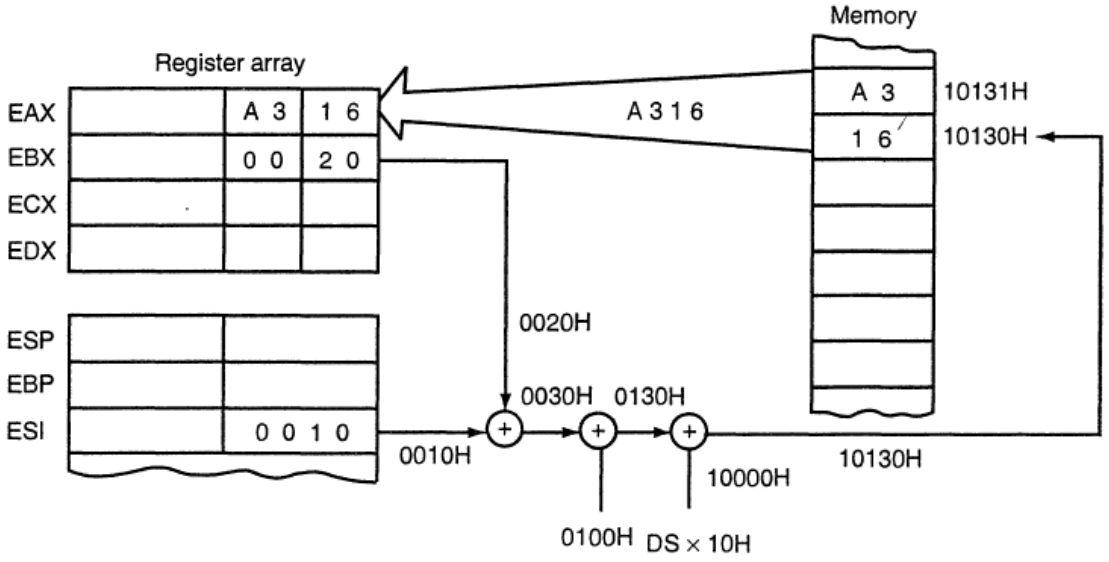


FIGURE 3–12 An example of base relative-plus-index addressing using a MOV AX,[BX+SI+100H] instruction. Note: DS = 1000H.

TABLE 3–8 Example base relative-plus-index instructions

Assembly Language	*Size*	*Operation*
MOV DH,[BX+DI+20H]	8-bits	Copies the byte contents of the data segment memory location addressed by the sum of BX, DI, and 20H into DH
MOV AX,FILE[BX+DI]	16-bits	Copies the word contents of the data segment memory location addressed by the sum of FILE, BX, and DI into AX
MOV LIST[BP+DI],CL	8-bits	Copies CL into the stack segment memory location addressed by the sum of LIST, BP, and DI
MOV LIST[BP+SI+4],DH	8-bits	Copies DH into the stack segment memory location addressed by the sum of LIST, BP, SI, and 4

Addressing Arrays with Base Relative-plus-index Addresssing. Suppose that a file of many records exists in memory and each record contains many elements. This displacement addresses the file, the base register addresses a record, and the index register addresses an element of a record. Figure 3–13 illustrates this very complex form of addressing.

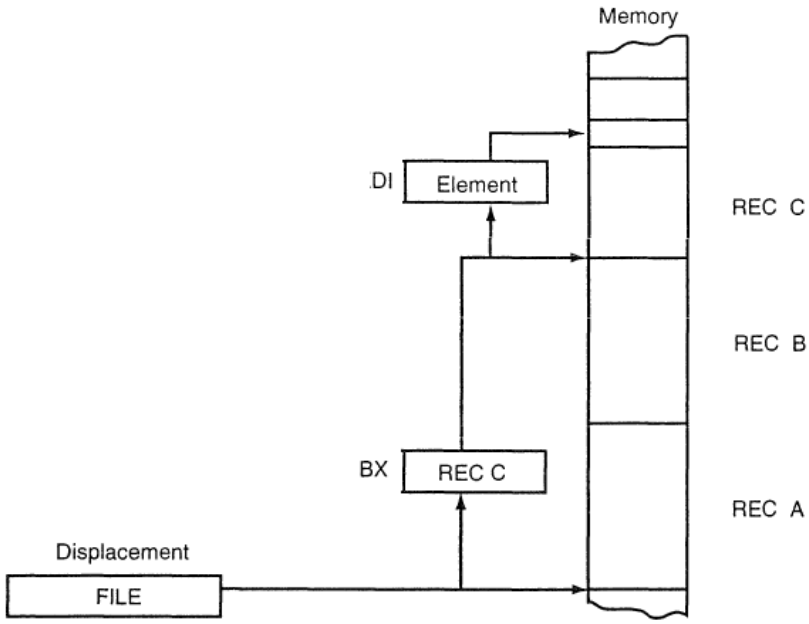


FIGURE 3–13 Base relative-plus-index addressing used to access a FILE that contains multiple records (REC)

PROGRAM MEMORY-ADDRESSING MODES

Program memory-addressing modes, used with the JMP and CALL instructions, consist of three distinct forms: direct, relative, and indirect. This section introduces these three addressing forms, using the JMP instruction to illustrate their operation.

Direct Program Memory Addressing

Direct program memory addressing is what many early microprocessors used for all jumps and calls. Direct program memory addressing is also used in high-level languages, such as the BASIC language GOTO and GOSUB instructions. The microprocessor uses this form of addressing, but not as often as relative and indirect program memory addressing are used.

The instructions for direct program memory addressing store the address with the opcode. For example, if a program jumps to memory location 10000H for the next instruction, the address (10000H) is stored following the opcode in the memory. Figure 3–14 shows the direct intersegment JMP instruction and the four bytes required to store the address 10000H. This JMP instruction loads CS with 1000H and IP with 0000H to jump to memory location 10000H for the next instruction. (An **intersegment jump** is a jump to any memory location within the entire memory system.) The direct jump is often called a *far jump* because it can jump to any memory location for the next instruction. In the real mode, a far jump accesses any location within the first 1M byte of memory by changing both CS and IP.

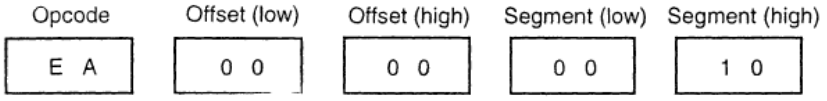


FIGURE 3–14 The 5-byte machine language version of a JMP [10000H] instruction

The only other instruction that uses direct program addressing is the intersegment or far CALL instruction. Usually, the name of a memory address, called a *label,* refers to the location that is called or jumped to instead of the actual numeric address. When using a label with the CALL or JMP instruction, most assemblers select the best form of program addressing.

Relative Program Memory Addressing

Relative program memory addressing is not available in all early microprocessors, but it is available to the Intel family of microprocessors. The term **relative** means "relative to the instruction pointer (IP)." For example, if a JMP instruction skips the next two bytes of memory, the address in relation to the instruction pointer is a 2 that adds to the instruction pointer. This develops the address of the next program instruction. An example of the relative JMP instruction is shown in Figure 3–15. Notice that the JMP instruction is a one-byte instruction with a one-byte or a two-byte displacement that adds to the instruction pointer. A one-byte displacement is used in short jumps, and a two-byte displacement is used in near jumps and calls. Both types are considered intrasegment jumps. (An **intrasegment jump** is a jump anywhere within the current code segment.)

Relative JMP and CALL instructions contain either an 8-bit or a 16-bit signed displacement that allows a forward memory reference or a reverse memory reference.

All assemblers automatically calculate the distance for the displacement and select the proper one-, two- or, four-byte form. If the distance is too far for a two-byte displacement in the 8086 through 80286 microprocessors, some assemblers use the direct jump. An 8-bit displacement *(short)* has a jump range of between +127 and −128 bytes from the next instruction, while a 16-bit displacement *(near)* has a range of ±32K bytes.

Indirect Program Memory Addressing

The microprocessor allows several forms of indirect program memory addressing for the JMP and CALL instructions. Table 3–10 lists some acceptable indirect program jump instructions, which can use any 16-bit register (AX, BX, CX, DX, SP, BP, DI, or SI), any relative register ([BP], [BX], [DI], or [SI]), and any relative register with a displacement.

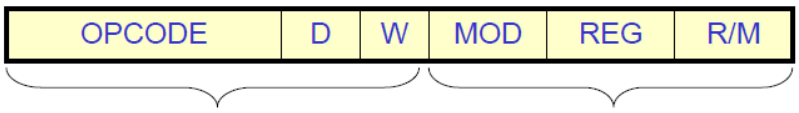
If a 16-bit register holds the address of a JMP instruction, the jump is near. For example, if the BX register contains a 1000H and a JMP BX instruction executes, the microprocessor jumps to offset address 1000H in the current code segment.

If a relative register holds the address, the jump is also considered an indirect jump. For example, a JMP [BX] instruction refers to memory location within the data segment at the offset address contained in BX. At this offset address is a 16-bit number that is used as the offset address in the intrasegment jump. This type of jump is sometimes called an *indirect-indirect* or *double-indirect jump.*

TABLE 3–10 Examples of indirect program memory addressing

Assembly Language	*Operation*
JMP AX	Jumps to the current code segment location addressed by the contents of AX
JMP CX	Jumps to the current code segment location addressed by the contents of CX
JMP NEAR PTR [BX]	Jumps to the current code segment location addressed by the contents of the data segment memory location addressed by BX
JMP NEAR PTR[DI+2]	Jumps to the current code segment location addressed by the contents of the data segment memory location addressed by DI plus 2
JMP TABLE[BX]	Jumps to the current code segment location addressed by the contents of the data segment memory location addressed by TABLE plus BX

Converting Assembly Language Instructions to Machine Code



An instruction can be coded with 1 to 6 bytes.

Byte 1 contains three kinds of information:

–Opcodefield (6 bits) specifies the operation such as add, subtract, or move

–Register Direction Bit (D bit)

It tells the register operand in REG field in byte 2 is source or destination operand:

–1:Data flow to the REG field from R/M

–0: Data flow from the REG field to the R/M

–Data Size Bit (W bit)

It specifies whether the operation will be performed on 8-bit or 16-bit data

–0: 8 bits

–1: 16 bits

Byte 2 has two fields:

–Mode field (MOD) –2 bits

–Register field (REG) -3 bits

–Register/memory field (R/M field) –2 bits

- REG field is used to identify the register for the first operand.

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

- 2-bit MOD field and 3-bit R/M field together specify the second operand

CODE	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M = 110, then 16-bit displacement follows

MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

Summary of addressing modes

Addressing Mode	Operand	Default Segment
Register	Reg	None
Immediate	Data	None
Direct	[offset]	DS
Register Indirect	[BX] [SI] [DI]	DS DS DS
Based Relative	[BX]+disp [BP]+disp	DS SS
Indexed Relative	[DI]+disp [SI]+disp	DS DS
Based Indexed Relative	[BX][SI or DI]+disp [BP][SI or DI]+disp	DS SS

16 bit Segment Assignments

Segment Registers	CS	DS	ES	SS
Offset Register	IP	SI,DI,BX	SI,DI,BX	SP,BP

Type of Memory Reference	Default Segment	Alternate Segment	Offset
Instruction Fetch	CS	none	IP
Stack Operations	SS	none	SP,BP
General Data	DS	CS,ES,SS	BX, address
String Source	DS	CS,ES,SS	SI, DI, address
String Destination	ES	None	DI