3.1 Data-Addressing Modes:

Program is a sequence of commands (instructions) used to tell a microcomputer what to do. Programs must always be coded in machine language (machine code) before they can be executed by the microprocessor. Machine code is encoded using 0s and 1s with single machine language instruction can take up one or more bytes of code. In assembly language, each instruction is described with alphanumeric symbols instead of 0s and 1s. Instruction can be divided into two parts: its ***opcode*** and ***operands***. Op-code identifies the operation that is to be performed and each opcode is assigned a unique letter combination called a ***mnemonic***. Operands describe the ***data*** that are to be processed as the microprocessor carried out, the operation specified by the opcode. For example, the move instruction is one of the instructions in the data transfer group of the 8086 instruction set. Execution of this instruction transfers a byte or a word of data from a ***source*** location to a ***destination*** location.

Because the MOV instruction is a very common and flexible instruction, 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. An **opcode**, or operation code, tells the microprocessor which operation to perform (i.e.; 8BC3 is the opcode for MOV AX,BX). We naturally assume that things move from left to right, whereas 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.

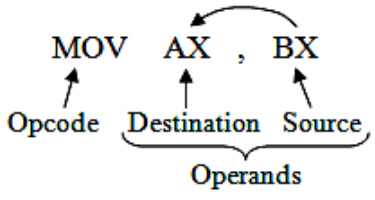
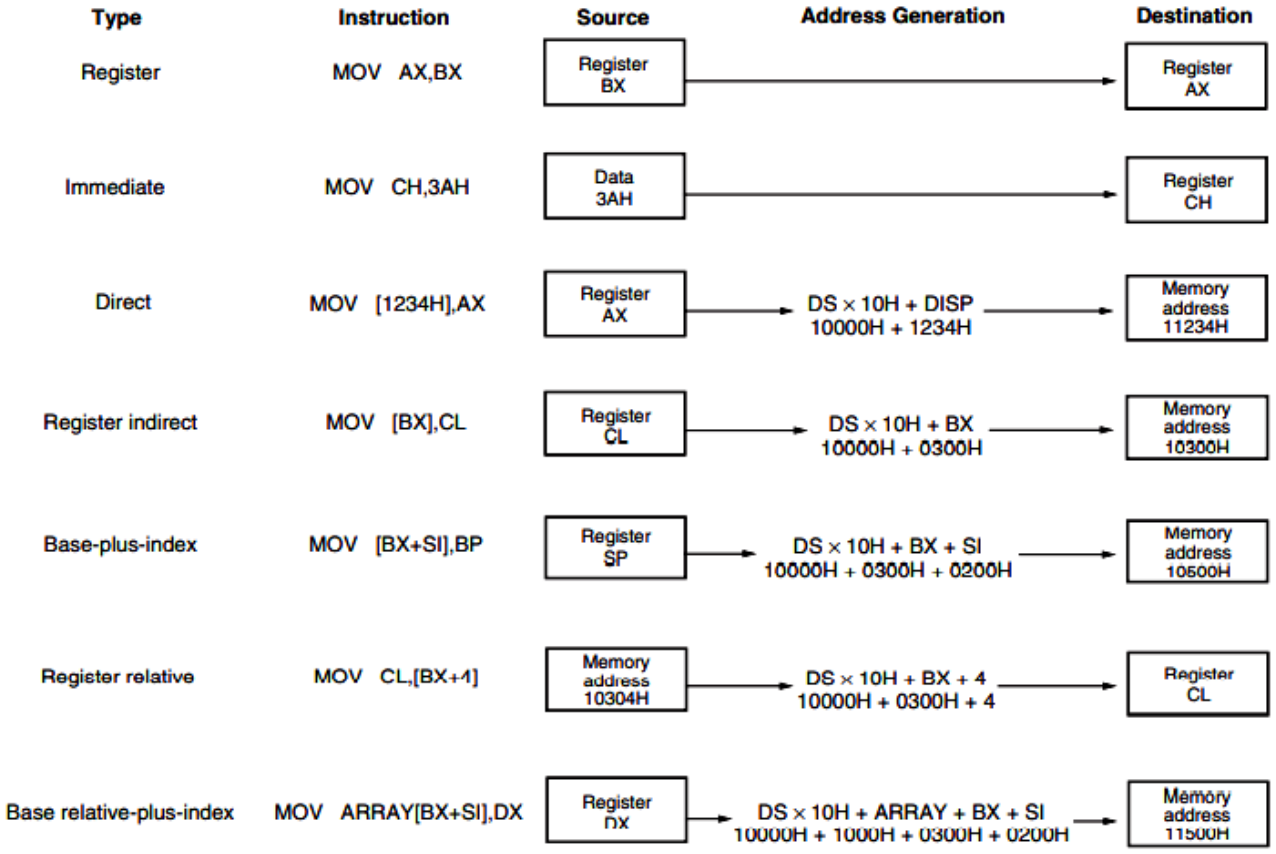


Fig. (3-1): The MOV instruction showing the source, destination, and direction of data flow.

In Figure 3–1, the MOV AX, BX instruction ***transfers (copies)*** the word contents of the source register (BX) into the destination register (AX). The source never changes, but the destination always changes. The MOV never actually picks up the data and ***moves*** it. Also, note the flag register remains ***unaffected*** by most data transfer instructions. The source and destination are often called **operands**.

Figure 3–2 shows all possible variations of the data-addressing modes using the MOV instruction. This illustration helps to show how each data-addressing mode is formulated with the MOV instruction and also serves as a reference on data-addressing modes.



Notes: BX = 0300H, SI = 0200H, ARRAY = 1000H, and DS = 1000H

Fig. (3-2): 8086–micrprocessor data-addressing modes.

The data-addressing modes are as follows:

a. Register addressing is the most common form of data addressing and, once the register names are learned, is the easiest to apply. Register addressing transfers a copy of a byte (8-bit: AH, AL, BH, BL, CH, CL, DH or DL) or word (16-bit: AX, BX, CX, DX, SI, DI, SP or BP) or segment registers (Sreg: CS, DS, ES, or SS) from the source register or contents of a memory location to the destination register or memory location. Note that: CS cannot be a destination. With register addressing, some MOV instructions and the PUSH and POP instructions. It is important for instructions to use registers that are the same size. Never mix an 8-bit register with a 16-bit register, because this is not allowed by the microprocessor and results in an error when assembled (i.e.; MOV AX, AL instruction are not allowed because the registers are of different sizes). It is also important to note that none of the MOV instructions affect the flag bits.

Table (3–1) Examples of register-addressed instructions.

Assembly Language	Size	Operation
MOV AL, BL	8 bit	Copies BL into AL
MOV CH, CL	8 bit	Copies CL into CH
MOV AX, BX	16 bit	Copies BX into AX
MOV AX, CX	16 bit	Copies CX into AX
MOV SP, BP	16 bit	Copies BP into SP
MOV DS, AX	16 bit	Copies AX into Segment Register DS
MOV SI, DI	16 bit	Copies DI into SI
MOV BX, ES	16 bit	Copies ES into BX
MOV DS, CX	16 bit	Copies CX into DS
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)

Note: There is no need to compute the effective address because the operand is in a register and no memory access involved.

Table 3–1 shows many variations (not all combinations) of register move instructions. For example, just the 8-bit subset of the MOV instruction has 64 different variations. A segment-to-segment register MOV instruction is about the only type of register MOV instruction not allowed. Note that the code segment register is not normally changed by a MOV instruction because the address of the next instruction is found by both IP 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.

Some ***rules*** in register addressing modes:

1. You may not specify CS as the destination operand.
   Example: **MOV CS, 02H –> wrong**
2. Only one of the operands can be a segment register. You cannot move data from one segment register to another with a single MOV instruction. To copy the value of CS to DS, you would have to use some sequence like:
   MOV DS,CS -> wrong
   MOV AX, CS
   MOV DS, AX -> the way we do it
3. You should never use the segment registers as data registers to hold arbitrary values. They should only contain segment addresses.

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 l234H from register CX into register BX. This erases the old contents (76AFH) of register BX, but the contents of CX remain unchanged.

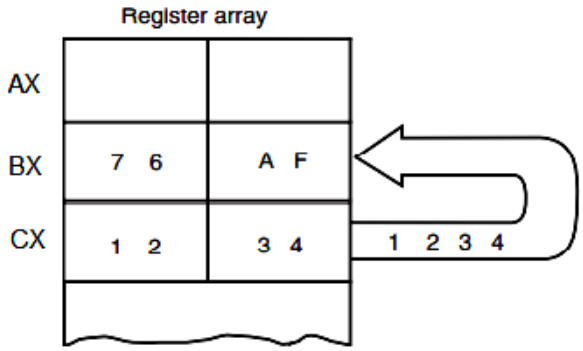


Fig. (3-3): The effect of executing the MOV BX, CX instruction at the point just before the BX register changes.

Example 3–1 shows a sequence of assembled instructions that copy various data between 8-, and 16-bit registers. As mentioned, the act of moving data from one register to another changes only the destination register, never the source. The last instruction in this example (MOV CS,AX) assembles without error, but causes problems if executed. If only the contents of CS change without changing IP, the next step in the program is unknown and therefore causes the program to go awry.

```
MOV AX,BX ;copy contents of BX into AX
MOV CL,DH ;copy contents of DH into CL
MOV CL,CH ;copy contents of CH into CL
MOV AX,CS ;copy CS into DS (two steps)
MOV DS,AX
MOV CS,AX ;copy AX into CS (causes problems)
```

b. 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, whereas the data transferred from a register or memory location are variable data. Immediate addressing (MOV instruction) transfers the source, an immediate byte, or word of data, into the destination register or memory location. The symbolic assembler portrays immediate data in many ways. The letter H appends hexadecimal data. If hexadecimal data begin with a letter, the assembler requires that the data start with a 0. For example, to represent a hexadecimal F2, 0F2H is used in assembly language. Decimal data are

represented as is and require no special codes or adjustments. (An example is the 100 decimal in the MOV AL,100 instruction). An ASCII-coded character or characters may be depicted in the immediate form if the ASCII data are enclosed in apostrophes. (An example is the MOV BH,‘A’ instruction, which moves an ASCII-coded letter A [41H] into register BH). Be careful to use the apostrophe (‘) for ASCII data and not the single quotation mark (‘). Binary data are represented if the binary number is followed by the letter B, or, in some assemblers, the letter Y. Table 3–2 shows many different variations of MOV instructions that apply immediate data.

Table 3–2 Examples of immediate addressing using the MOV instruction.

Assembly Language	Size	Operation
MOV AL, 20	8 bit	Copies 20 decimal (14H) into register AL
MOV BL, 44	8 bit	Copies 44 decimal (2CH) into register BL
MOV AX, 44H	16 bit	Copies 0044H into register AX
MOV BX, 55H	16 bit	Copies a 0055H into register BX
MOV SI, 0	16 bit	Copies 0000H into register SI
MOV CH, 100	8 bit	Copies 100 decimal (64H) into register CH
MOV AL, 'A'	8 bit	Copies ASCII of A into register AL
MOV AH, 1	8 bit	Copies 1 decimal (01H) into register AH
MOV AX, 'AB'	16 bit	Copies ASCII of AB into register AX
MOV DX, 'Ahmed'	16 bit	Copies an ASCII Ahmed into register DX
MOV CL, 11001110B	8 bit	Copies 11001110 binary into register CL

Figure 3–4 shows the operation of a MOV AX,3456H instruction. This instruction copies the 3456H from the instruction, located in the memory immediately following the hexadecimal opcode, into register AX. As with the MOV instruction illustrated in Figure 3–3, the source data overwrites the destination data.

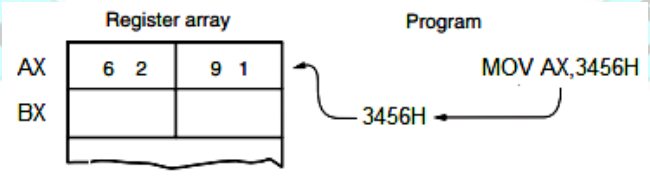


Fig. (3-4): The operation of the MOV AX,3456H instruction. This instruction copies the immediate data (3456H) into AX.

Example 3–2 shows various immediate instructions in a short assembly language program that places 0000H into the 16-bit registers AX, BX, and CX. This is followed by instructions that use register addressing to copy the contents of AX into registers SI, DI, and BP.

```
MOV AX,0 ;place 0000H into AX
MOV BX,0 ;place 0000H into BX
```

```
MOV CX,0 ;place 0000H into CX
MOV SI,AX ;copy AX into SI
MOV DI,AX ;copy AX into DI
MOV BP,AX ;copy AX into BP
```

Note: A comment always begins with a semicolon (;).

EXAMPLE 3–3

```
DATA1=23H, and DATA2=1000H
MOV AL,BL ;copy BL into AL
MOV BH,AL ;copy AL into BH
MOV CX,200 ;copy 200 into C
```

c. Direct addressing Most instructions in typical program can use the direct data-addressing mode. There are two basic forms of direct data addressing: (1) ***direct addressing***, which applies to a MOV between a memory location and AL, AX, and (2) ***displacement addressing***, which applies to almost any instruction in the instruction set. In either case, the address is formed by adding the displacement to the default data segment address or an alternate segment address.

Direct addressing: moves a byte or word between a memory location (located within the data segment) and a register (8-bit: AL, or l6-bit: AX). The instruction set does not support a memory-to-memory transfer, except with the MOVS instruction. A MOV instruction using this type of addressing is usually a 3-byte long instruction.

Table 3–3 lists the direct-addressed instructions. All other instructions that move data from a memory location to a register, called displacement addressed instructions, require 4 or more bytes of memory for storage in a program.

Table 3–3 Direct addressed instructions using AX, and AL.

Assembly Language	Size	Operation
MOV AL, NUMBER	8 bit	Copies the byte contents of data segment memory location NUMBER into register AL
MOV AX, COW	16 bit	Copies the word contents of data segment memory location COW into register AX
MOV NEWS, AL	8 bit	Copies AL into byte memory location NEWS
MOV THERE, AX	16 bit	Copies AX into word memory location THERE
MOV ES:[2000H], AL	8 bit	Copies AL into extra segment memory at offset address 2000H
MOV AL, MOUSE	8 bit	Copies the contents of location MOUSE into

		AL; in 16-bit mode MOUSE can be any address
MOV AL, DS:[2000H] Or MOV AL, [2000H]	8 bit	Copies the contents of the memory location with offset 2000H into register AL
MOV AX, DS:[8088H] Or MOV AX, [8088H]	16 bit	Copies the contents of the memory location with offset 8088H & 8089H into register AX
MOV DS:[1234H], DL Or MOV [1234H], DL	8 bit	Store the value in the DL register to memory location with offset 1234H
MOV DS:[1234H], DX Or MOV [1234H], DX	16 bit	Store the value in the DL register to memory location with offset 1234H & DH register to memory location with offset 1235H

The MOV AL,DATA instruction, as represented by most assemblers, loads AL from the 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]7 instruction. 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) and 10000H (the data segment address of 1000H times 10H) in a system operating in the real mode.

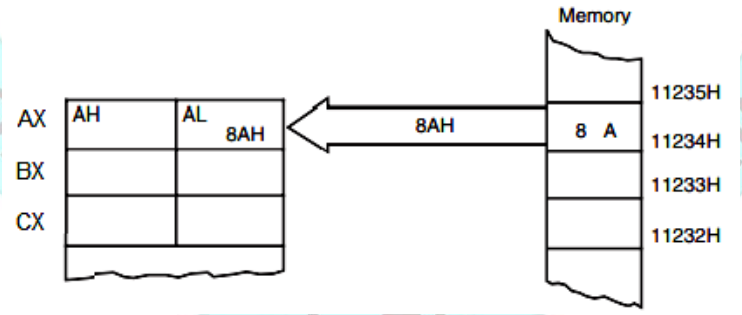


Fig. (3-5): The operation of the MOV AL,[1234H] instruction when . DS = 1000.

Displacement Addressing: Displacement addressing is almost identical to direct addressing, except that the instruction is 4 bytes wide instead of 3. This type of direct data addressing is much more flexible because most instructions use it.

If the operation of the MOV CL,DS:[1234H] instruction is compared to that of the MOV AL,DS:[1234H] instruction of Figure 3–5, we see that both basically perform the same operation except for the destination

register (CL versus AL). Another difference only becomes apparent upon examining the assembled versions of these two instructions. The MOV AL,DS:[1234H] instruction is 3 bytes long and the MOV CL,DS:[1234H] instruction is 4 bytes long, as illustrated in Example 3–4. This example shows how the assembler converts these two instructions into hexadecimal machine language. You must include the segment register DS: in this example, before the [offset] part of the instruction. You may use any segment register, but in most cases, data are stored in the data segment, so this example uses DS:[1234H].

EXAMPLE 3–4

```
MOV AL,DS:[1234H]
MOV CL,DS:[1234H]
```

Table 3–4 lists some MOV instructions using the displacement form of direct addressing. Not all variations are listed because there are many MOV instructions of this type. The segment registers can be stored or loaded from memory.

Table 3–4 Examples of direct data addressing using a displacement.

Assembly Language	Size	Operation
MOV CH, DOG	8 bit	Copies the byte contents of data segment memory location DOG into register CH
MOV CH, DS:[1000H]	8 bit	Copies the byte contents of data segment memory offset address 1000H into register CH
MOV ES, DATA6	16 bit	Copies the word contents of data segment memory location DATA6 into register ES
MOV DATA7, BP	16 bit	Copies BP into data segment memory location DATA7
MOV NUMBER, SP	16 bit	Copies SP into data segment memory location NUMBER

EXAMPLE 3–5

```
DATA1=10H, DATA2=00H, DATA3=0000H, and DATA4=AAAAH
MOV AL,DATA1 ;copy DATA1 into AL
MOV AH,DATA2 ;copy DATA2 into AH
MOV DATA3,AX ;copy AX into DATA3
MOV BX,DATA4 ;copy DATA4 into BX
```

d. Register indirect addressing transfers a byte or word between a register and a memory location through an offset address held by an index or base registers, these are: BP, BX, DI, and S1.

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 the BP register addresses memory, the stack segment is used by default. These settings are considered the default for these four index and base registers. For example, the MOV AL,[DI] instruction is clearly a byte-sized move instruction, but the MOV [DI],10H instruction is ambiguous. Does the MOV [DI],10H instruction address a byte-, or word-sized memory location? The assembler can't determine the size of the 10H. The instruction MOV [DI],10H clearly designates the location addressed by DI as a byte-sized memory location.

Table 3–5 Examples of register indirect addressing.

Assembly Language	Size	Operation
MOV AL, [BX]	8 bit	Copies the contents of the memory location with offset BX into register AL
MOV AX, [BX]	16 bit	Copies the word-sized contents of the memory location with offset BX & BX+1 into register AX
MOV CX, [BX]	16 bit	Copies the word contents of data segment memory location addressed by BX into CX
MOV AL, [BP]	8 bit	Copies the contents of the memory location with offset BP into register AL
MOV AL, [SI]	8 bit	Copies the contents of the memory location with offset SI into register AL
MOV AL, [DI]	8 bit	Copies the contents of the memory location with offset DI into register AL
MOV [BP], DL*	8 bit	Copies DL into the stack segment memory location addressed by BP
MOV [DI], BH	8 bit	Copies BH into the data segment memory location addressed by DI
MOV [DI], [BX]	-	Memory-to-memory transfers are not allowed except with string instructions
MOV AL, [DX]	8 bit	Copies the byte contents of the data segment memory location addressed by DX into AL

***Note: Data addressed by BP are by default in the stack segment, while other indirect addressed instructions use the data segment by default.**

The [BX], [SI], and [DI] modes use the DS segment by default (1000H). The [BP] addressing mode uses the stack segment (SS) by default (3400H). You can use the segment override prefix symbols if you wish to access data in different segments. The following instructions demonstrate the use of these overrides:

```
MOV AL, CS:[BX]
```

MOV AL, DS:[BP]
MOV AL, SS:[SI]
MOV AL, ES:[DI]

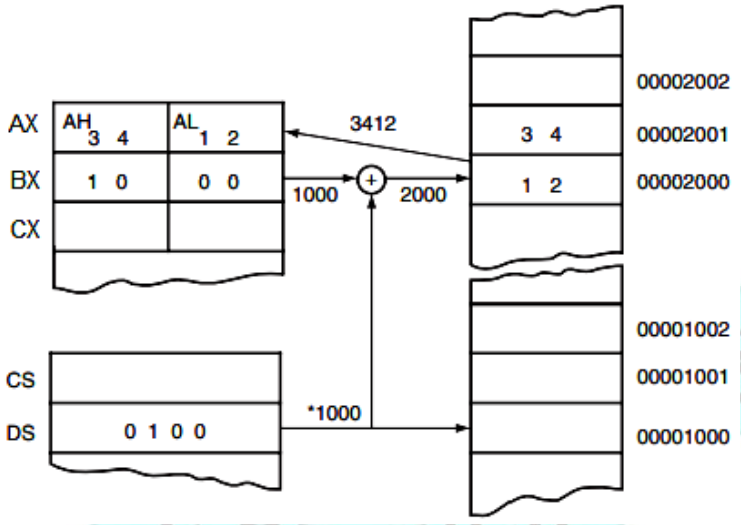
For example:

MOV SI, 1234H
MOV AL, [SI]

If SI contains 1234H and DS contains 0200H the result produced by executing the instruction is that the contents of the memory location at address:

PA = 02000H + 1234H = 03234 are moved to the AL register.

For example, if register BX contains 1000H and the MOV AX,[BX] instruction executes, the word contents of data segment offset address 1000H are 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. In addition to using the BP, BX, DI, and SI registers to indirectly address memory. Some typical instructions using indirect addressing appear in Table 3–5.



*After DS is appended with a 0.

Fig. (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.

The sequence shown in Example 3–7 loads register BX with the starting address of the table and it initializes the count, located in register CX, to 50. The OFFSET directive tells the assembler to load BX with the offset address of memory location TABLE, not the contents of TABLE. For example, the MOV BX,DATAS instruction copies the contents of memory location DATAS into BX, while the MOV BX,OFFSET

DATAS instruction copies the offset address DATAS into BX. When the OFFSET directive is used with the MOV instruction, the assembler calculates the offset address and then uses a MOV immediate instruction to load the address in the specified 16-bit register.

EXAMPLE 3–7

```
MOV AX,0
MOV ES,AX ;address segment 0000 with ES
MOV BX,OFFSET DATAS ;address DATAS array with BX
MOV CX,50 ;load counter with 50
AGAIN:
MOV AX,ES:[046CH] ;get clock value
MOV [BX],AX ;save clock value in DATAS
INC BX ;increment BX to next element
INC BX
LOOP AGAIN ;repeat 50 times
```

Once the counter and pointer are initialized, a repeat-until CX = 0 loop executes. Here data are read from extra segment memory location 46CH with the MOV AX,ES:[046CH] instruction and stored in memory that is indirectly addressed by the offset address located in register BX. Next, BX is incremented (1 is added to BX) twice to address the next word in the table. Finally, the LOOP instruction repeats the LOOP 50 times. The LOOP instruction decrements (subtracts 1 from) the counter (CX); if CX is not zero, LOOP causes a jump to memory location AGAIN. If CX becomes zero, no jump occurs and this sequence of instructions ends. This example copies the most recent 50 values from the clock into the memory array DATAS. This program will often show the same data in each location because the contents of the clock are changed only 18.2 times per second.

e. Base-plus-index addressing is similar to indirect 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). The base register often holds the beginning location of a memory array, whereas 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 3–7 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.

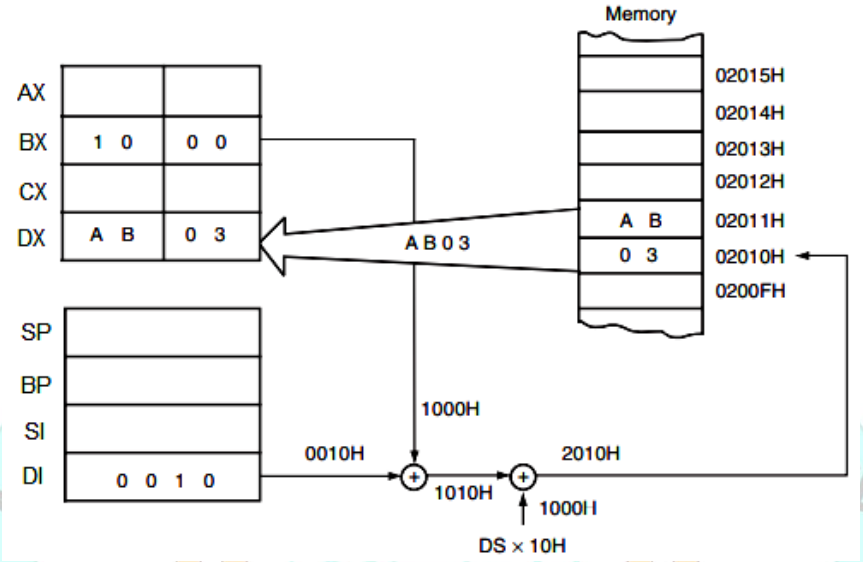


Fig. (3-7): An example showing how the base-plus-index addressing mode functions for the MOV DX,[BX+DI] instruction. Notice that memory address 02010H is accessed because . DS = 0100H, BX = 100H, and DI=0010H.

Table 3–6 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.

Table 3–6 Examples of base-plus-index addressing.

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

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 3–8 shows the use of BX and DI to access an element in an array of data.

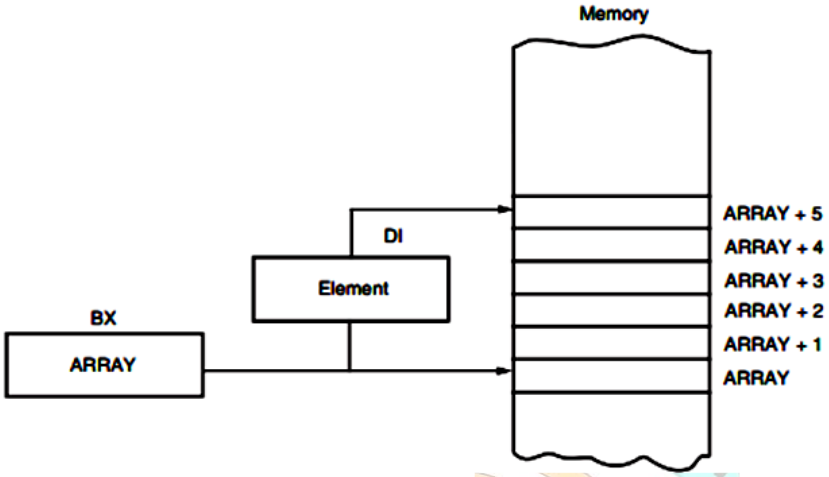


Fig. (3-8): An example of the base-plus-index addressing mode. Here an element (DI) of an ARRAY (BX) is addressed.

A short program, listed in Example 3–8, moves array element 10H into array element 20H. Notice that the array element number, loaded into the DI register, addresses the array element. Also notice how the contents of the ARRAY have been initialized so that element 10H contains 29H.

EXAMPLE 3–8

```
ARRAY=16 ;setup ARRAY element 10H
MOV BX,ARRAY ;address ARRAY
MOV DI,10H ;address element
MOV AL,[BX+DI] ;get element 10H
MOV DI,20H ;address element 20H
MOV [BX+DI],AL ;save in element 20H
```

f. Register relative addressing is similar to base-plus-index addressing moves a byte or word between a register and the memory location addressed by an index or base register plus a displacement. 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). (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).

Figure 3–9 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×0H, BX, and the displacement of 1000H, which addresses location 03100H. Remember that BX, DI, or SI addresses the data segment and BP addresses the stack segment. Table 3–7 lists a few instructions that use register relative addressing.

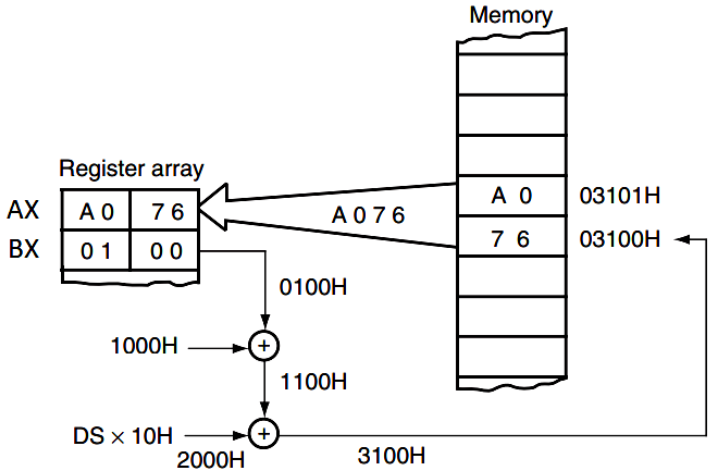


Fig. (3-9): The operation of the MOV AX, [BX+1000H] instruction, when BX = 0100H and DS = 0200H.

The displacement is a number added to the register within the [ ], as in the MOV AL,[DI+2] instruction, or it can be a displacement is subtracted from the register, as in MOV AL,[SI–l]. 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 the MOV AL,DATA[DI+3] instruction.

Table 3–7 Examples of register relative addressing.

Assembly Language	Size	Operation
MOV AX, [DI+100H]	16 bit	Copies the word contents of data segment memory location addressed by DI+100H into AX
MOV ARRAY[SI], BL	8 bit	Copies BL into data segment memory location addressed by ARRAY+SI
MOV LIST[SI+2], CL	8 bit	Copies CL into the data segment memory location addressed by LIST+SI+2
MOV DI, SET_IT[BX]	16 bit	Copies the word contents of data segment memory location addressed by SET_IT+BX into DI
MOV DI, [AX+10H]	16 bit	Copies the word contents of data segment memory location addressed by AX+10H into DI

Addressing Array Data with Register Relative. It is possible to address array data with register relative addressing, such as one does with base-plus-index addressing. In Figure 3–10, 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.

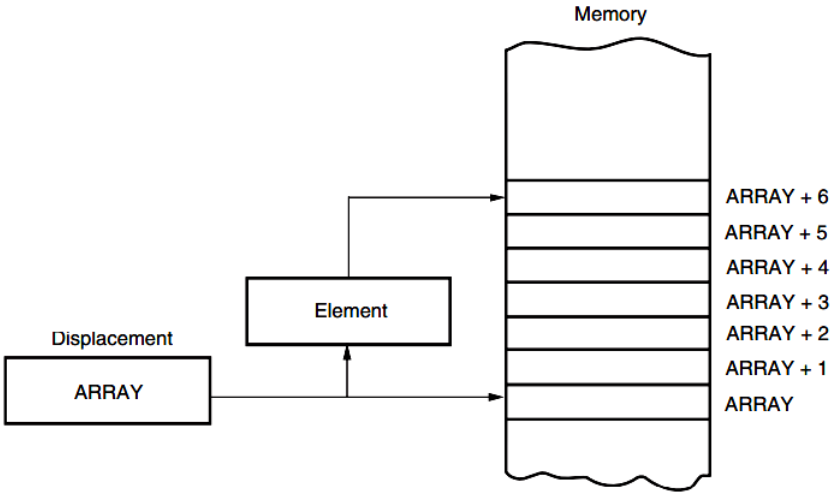


Fig. (3-10): Register relative addressing used to address an element of ARRAY. The displacement addresses the start of ARRAY, and DI accesses an element.

Example 3–9 shows how this new addressing mode can transfer the contents of array element 10H into array element 20H. Notice the similarity between this example and Example 3–8. The main difference is that, in Example 3–9, register BX is not used to address memory ARRAY; instead, ARRAY is used as a displacement to accomplish the same task.

EXAMPLE 3–9

```
ARRAY=16 ;setup ARRAY element 10H
MOV DI,10H ;address element
MOV AL,ARRAY [DI] ;get array element 10H
MOV DI,20H ;address element 20H
MOV ARRAY [DI],AL ;save in element 20H
```

g. Base relative-plus-index addressing is similar to base-plus-index addressing but it adds a displacement, besides using a base register and an index register transfers a byte or word between a register and the memory location addressed by a base and an index register plus a displacement. This type of addressing mode often addresses a two-dimensional array of memory data. (Example: MOV AX, ARRAY[BX+DI] or MOV AX, [BX+DI+4]. These instructions load AX from a data segment memory location. The first instruction uses an address formed by adding ARRAY, BX, and DI and the second by adding BX, DI, and 4).

Addressing Data with Base Relative-Plus-Index. Base relative-plus-index addressing is the least-used addressing mode. Figure 3–11 shows how data are referenced if the instruction executed by the microprocessor

is 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=0100H, 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 programming.

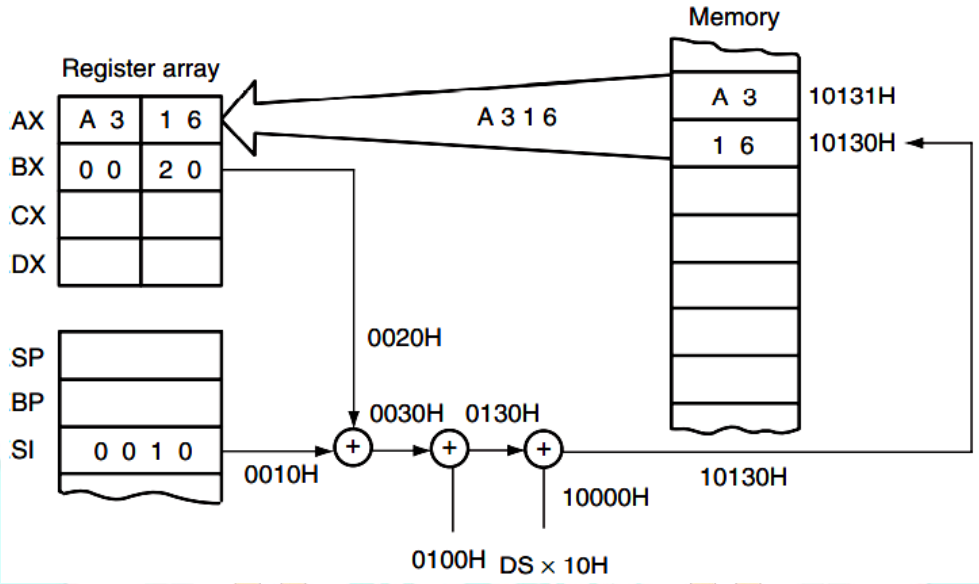


Fig. (3-11): An example of base relative-plus-index addressing using a MOV AX,[BX+SI+100H] instruction. Note: DS = 1000H.

Addressing Arrays with Base Relative-Plus-Index. Suppose that a file of many records exists in memory and each record contains many elements. The displacement addresses the file, the base register addresses a record, and the index register addresses an element of a record. Figure 3–12 illustrates this very complex form of addressing.

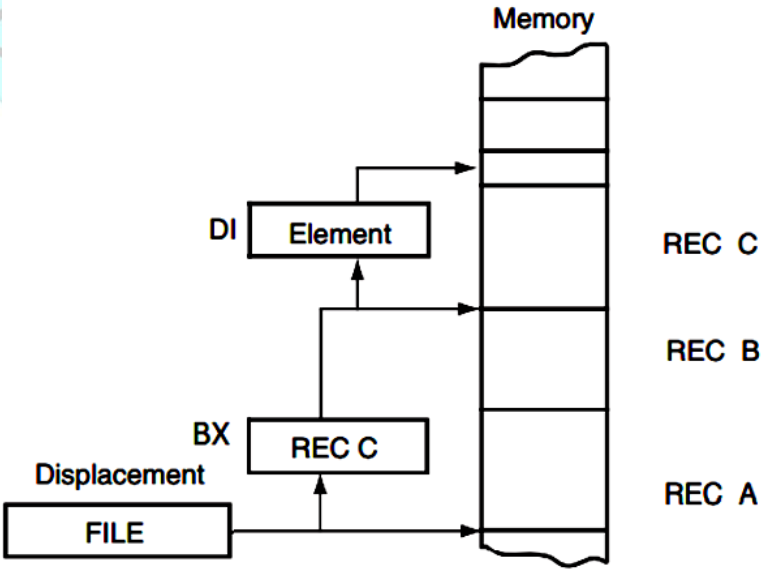


Fig. (3-12): Base relative- plus-index addressing used to access a FILE that contains multiple records (REC).

Example 3–10 provides a program that copies element 0 of record A into element 2 of record C by using the base relative-plus-index mode of addressing. This example FILE contains four records and each record contains 10 elements. Notice how the THIS BYTE statement is used to define the label FILE and RECA as the same memory location.

EXAMPLE 3–10

```
RECA=10 ; 10 bytes for record A
RECB=10 ; 10 bytes for record B
RECC=10 ; 10 bytes for record C
RECD=10 ; 10 bytes for record D
MOV BX,OFFSET RECA ;address record A
MOV DI,0 ;address element 0
MOV AL,FILE[BX+DI] ;get data
MOV BX,OFFSET RECC ;address record C
MOV DI,2 ;address element 2
MOV FILE[BX+DI],AL ;save data
```

3.2 Program Memory-Addressing Modes:

Program memory-addressing modes, used with the JMP (jump) 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.

3.2.1 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 3–13 shows the direct intersegment JMP instruction and the 4 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. In protected mode operation, the far jump accesses a new code segment descriptor from the descriptor table, allowing it to jump to any memory location.

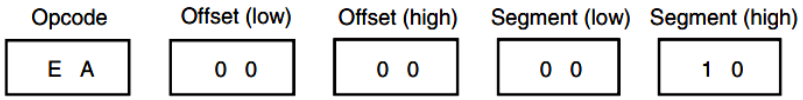


Fig. (3-13): The 5-byte machine language version of a JMP [10000H] instruction.

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.

3.2.2 Relative Program Memory Addressing:

Relative program memory addressing is not available in all early microprocessors, but it is available to this family of microprocessors. The term relative means "relative to the instruction pointer (IP)." For example, if a JMP instruction skips the next 2 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 3–14. Notice that the JMP instruction is a 1-byte instruction, with a 1-byte or a 2-byte displacement that adds to the instruction pointer. A 1-byte displacement is used in short jumps, and a 2-byte displacement is used with near jumps and calls. Both types are considered to be intrasegment jumps. (An ***intrasegment jump*** is a jump anywhere within the current code segment).

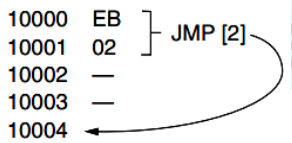


Fig. (3-14): A JMP [2] instruction which skips over the 2 bytes of memory that follow the JMP instruction.

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 1-, or 2-byte form. If the distance is too far for a 2-byte displacement in an 8086 microprocessor, some assemblers use the direct jump. An 8-bit displacement (short) has a jump

range of between and bytes from the next instruction; a 16-bit displacement (near) has a range of bytes.

3.2.3 Indirect Program Memory Addressing:

The microprocessor allows several forms of program indirect memory addressing for the JMP and CALL instructions. Table 3–8 lists some acceptable program indirect 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 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 to be an indirect jump. For example, JMP [BX] refers to the memory location within the data segment at the offset address contained in BX. At this offset address is a l6-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 3–7 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 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 address by TABLE plus BX

Figure 3–15 shows a jump table that is stored, beginning at memory location TABLE. This jump table is referenced by the short program of Example 3–11. In this example, the BX register is loaded with a 4 so, when it combines in the JMP TABLE[BX] instruction with TABLE, the effective address is the contents of the second entry in the 16-bit-wide jump table.

```
TABLE  DW  LOC0
       DW  LOC1
       DW  LOC2
       DW  LOC3
```

Fig. (3-15): A jump table that stores addresses of various programs. The exact address chosen from the TABLE is determined by an index stored with the jump instruction.

EXAMPLE 3–11

```
MOV BX,4 ;address LOC2
JMP TABLE[BX] ;jump to LOC2
```

3.3 Stack Memory-Addressing Modes:

The stack plays an important role in microprocessors. It holds data temporarily and stores the return addresses used by procedures. The stack memory is an LIFO (last-in, first-out) memory, which describes the way that data are stored and removed from the stack. Data are placed onto the stack with a PUSH instruction and removed with a POP instruction. The CALL instruction also uses the stack to hold the return address for procedures and a RET (return) instruction to remove the return address from the stack.

The stack memory is maintained by two registers: the stack pointer (SP) and the stack segment register (SS). Whenever a word of data is pushed onto the stack [see Figure 3–16(a)], the high-order 8 bits are placed in the location addressed by SP–1. The low-order 8 bits are placed in the location addressed by SP–2. The SP is then decremented by 2 so that the next word of data is stored in the next available stack memory location. The SP register always points to an area of memory located within the stack segment. The SP register adds to SS×10H to form the stack memory address in the real mode. In protected mode operation, the SS register holds a selector that accesses a descriptor for the base address of the stack segment.

Whenever data are popped from the stack [see Figure 3–16(b)], the low-order 8 bits are removed from the location addressed by SP. The high-order 8 bits are removed from the location addressed by SP+1. The SP register is then incremented by 2. Table 3–11 lists some of the PUSH and POP instructions available to the microprocessor. Note that PUSH and POP store or retrieve words of data—never bytes—in the 8086 microprocessors. Data may be pushed onto the stack from any 16-bit register or segment register. Data may be popped off the stack into any register or any segment register except CS. The reason that data may not be popped from the stack into CS is that this only changes part of the address of the next instruction.

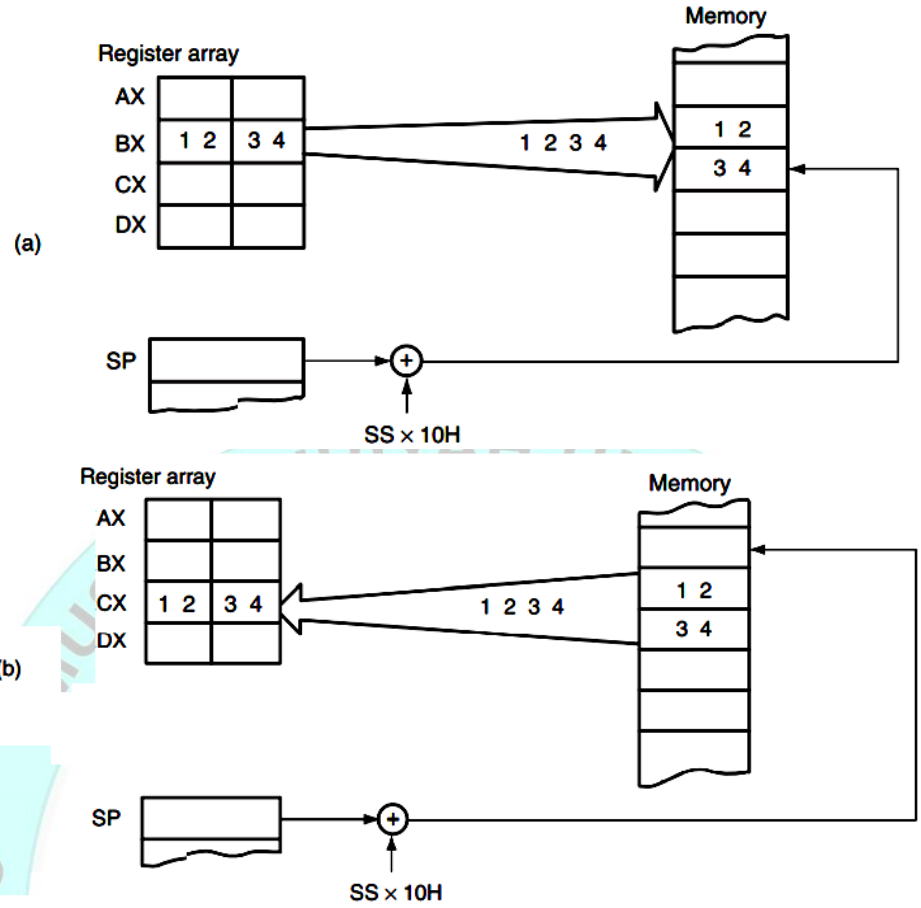


Fig. (3-16): The PUSH and POP instructions: (a) PUSH BX places the contents of BX onto the stack; (b) POP CX removes data from the stack and places them into CX. Both instructions are shown after execution.

Note the examples in Table 3–8, which show the order of the registers transferred by the PUSHA and POPA instructions.

Table 3–8 Example PUSH and POP instructions.

Assembly Language	*Operation*
POPF	Removes a word from the stack and places it into the flag register
POPFD	Removes a doubleword from the stack and places it into the EFLAG register
PUSHF	Copies the flag register to the stack
PUSHFD	Copies the EFLAG register to the stack
PUSH AX	Copies the AX register to the stack
POP BX	Removes a word from the stack and places it into the BX register
PUSH DS	Copies the DS register to the stack
PUSH 1234H	Copies a word-sized 1234H to the stack
POP CS	This instruction is illegal
PUSH WORD PTR[BX]	Copies the word contents of the data segment memory location addressed by BX onto the stack
PUSHA	Copies AX, CX, DX, BX, SP, BP, DI, and SI to the stack
POPA	Removes the word contents for the following registers from the stack: SI, DI, BP, SP, BX, DX, CX, and AX

Example 3–12 lists a short program that pushes the contents of AX, BX, and CX onto the stack. The first POP retrieves the value that was pushed onto the stack from CX and places it into AX. The second POP places the original value of BX into CX. The last POP places the value of AX into BX.

EXAMPLE 3–12

```
MOV AX,1000H ;load test data
MOV BX,2000H
MOV CX,3000H
PUSH AX ;1000H to stack        POP AX ;3000H to AX
PUSH BX ;2000H to stack       POP CX ;2000H to CX
PUSH CX ;3000H to stack       POP BX ;1000H to BX
```

Question: Compute the physical address for the specified operand in each of the following instructions. The register contents and variable are as follows: (CS)=0A00H, (DS)=0B00H, (SS)=0D00H, (SI)=0FF0H, (DI)=00B0H, (BP)=00EAH and (IP)=0000H, LIST=00F0H, AX=4020H, BX=2500H.

1) Destination operand of the instruction MOV LIST [BP+DI] , AX

2) Source operand of the instruction MOV CL , [BX+200H]

3) Destination operand of the instruction MOV [DI+6400H] , DX

4) Source operand of the instruction MOV AL, [BP+SI-400H]

5) Destination operand of the instruction MOV [DI+BP] , AX

6) Source operand of the instruction MOV CL , [BP+200H]

7) Destination operand of the instruction MOV [BX+DI+6400H] , CX

8) Source operand of the instruction MOV AL , [BP- 0200H]

9) Destination operand of the instruction MOV [SI] , AX

10) Destination operand of the instruction MOV [BX][DI]+0400H,AL

11) Source operand of the instruction MOV AX, [BP+200H]

12) Source operand of the instruction MOV AL, [SI-0100H]

13) Destination operand of the instruction MOV DI,[SI]

14) Destination operand of the instruction MOV [DI]+CF00H,AH

15) Source operand of the instruction MOV CL, LIST[BX+200H]